DOCUMENT RESUME

ED 301 176 IR 013 516

AUTHOR Clancey, William J.
TITLE Intelligent Tutoring Systems: A Tutorial Survey.
INSTITUTION Stanford Univ., Calif. Dept. of Computer Science.
SPONS AGENCY Office of Naval Research, Arlington, Va. Personnel and Training Research Programs Office.
REPORT NO KSL-86-58; ONR-TR-22; STAN-CS-87-1174
PUB DATE Sep 86
NOTE 58p.; For a related report, see IR 013 515.
PUB TYPE Information Analyses (070) -- Viewpoints (120)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Simulation; Computer System Design; *Expert Systems; Instructional Effectiveness; Learning Theories; *Man Machine Systems; *Models; Problem Solving
IDENTIFIERS *Intelligent Tutoring Systems; Knowledge Representation; *Learner Control

ABSTRACT

This survey of intelligent tutoring systems describes the components of these systems, different teaching scenarios, and the relation of these systems to a theory of instruction. It argues that the underlying pedagogical approach is to make latent knowledge manifest by using different forms of quantitative modeling: (1) simulating physical processes; (2) simulating expert problem solving, including strategies for monitoring and controlling problem solving (metacognition); (3) modeling the plans behind procedural behavior; and (4) forcing articulation of model inconsistencies through the Socratic method of instruction. Proceeding chronologically, examples of intelligent tutoring systems are described in terms of their internal knowledge representations and the evolving pedagogical theory. It also argues that, although these programs are generally only research projects, examples of what they can do make abundantly clear the long term scientific and software-engineering advantages of the new modelling methodology. The text is supplemented by 13 figures, and 31 references are provided. (Author/EW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED301176

September 1986

Report No. STAN-CS-87-1174
Also numbered KSL-86-58

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☑ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Intelligent Tutoring Systems: A Tutorial Survey

by

William J. Clancey

Department of Computer Science

Stanford University
Stanford, CA 94305

IR013516

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-87-1174 or KSL-86-58	5. MONITORING ORGANIZATION REPORT NUMBER(S) ONR TECHNICAL REPORT # 22
6a. NAME OF PERFORMING ORGANIZATION STANFORD KNOWLEDGE SYSTEMS LABORATORY 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION PERSONNEL AND TRAINING RESEARCH PROGRAMS
6c. ADDRESS (City, State, and ZIP Code) COMPUTER SCIENCE DEPARTMENT 701 WELCH ROAD, BUILDING C PALO ALTO, CA 94304	7b. ADDRESS (City, State, and ZIP Code) OFFICE OF NAVAL RESEARCH (CODE 1142PT) 800 NORTH QUINCY STREET ARLINGTON, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85K-0305
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
61153N	RR04206	0C	NR702-003

11. TITLE (Include Security Classification)
INTELLIGENT TUTORING SYSTEMS: A TUTORIAL SURVEY

12. PERSONAL AUTHOR(S)
WILLIAM J. CLANCEY

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
TECHNICAL	FROM _____ TO _____	SEPTEMBER 1986	56

16. SUPPLEMENTARY NOTATION
KSL #86-58; TO APPEAR IN CURRENT ISSUES IN EXPERT SYSTEMS (P. DUFOUR AND A VAN LAMSWEERDE, EDITORS) ACADEMIC PRESS, 1987, PAGES 39-78.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
05	09	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
INTELLIGENT TUTORING SYSTEMS, PEDAGOGY, KNOWLEDGE REPRESENTAION, MAN-MACHINE INTERFACE, USER MODEL

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This survey of Intelligent Tutoring Systems is based on a tutorial originally presented by John Seely Brown, Richard R. Burton (Xerox-Parc, USA) and William J. Clancey at the National Conference on AI (AAAI) in Austin, Texas in August, 1984. The survey describes the components of tutoring systems, different teaching scenarios, and their relation to a theory of instruction. The underlying pedagogical approach is to make latent knowledge manifest, which the research accomplishes by different forms of qualitative modeling: simulating physical processes; simulating expert problem-solving, including strategies for monitoring and controlling problem solving (metacognition); modeling the plans behind procedural behavior; and forcing articulation of model inconsistencies through the Socratic method of instruction. Proceeding chronologically, examples of intelligent tutoring systems are described in terms of their internal knowledge representations and the evolving pedagogical theory. Although these programs are generally only research projects, examples of what they can do make abundantly clear the long-term scientific and software-engineering advantages of the new modeling methodology.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
DR. SUSAN CHIPMAN	(202) 696-4318	ONR 1142PT

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

INTELLIGENT TUTORING SYSTEMS:

A TUTORIAL SURVEY

by

William J. Clancey

Stanford Knowledge Systems Laboratory
Department of Computer Science
701 Welch Road, Building C
Palo Alto, CA 94304

The studies reported here were supported (in part) by:

The Office of Naval Research
Personnel and Training Research Programs
Psychological Sciences Division
Contract No. N00014-85K-0305

The Josiah Macy, Jr. Foundation
Grant No. B852005
New York City

The views and conclusions contained in this document are the authors' and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Office of Naval Research or the U.S. Government.

Approved for public release: distribution unlimited. Reproduction in whole or in part is permited for any purpose of the United States Government.

Table of Contents

Abstract 1
1. Introduction 1
    1.1. Components of an ITS 2
    1.2. Teaching Scenarios 6
    1.3. Learning scenarios for an intelligent tutor 8
    1.4. A naive theory of coaching 9
    1.5. Making latent knowledge manifest 11
2. Survey of Specific Systems 12
    2.1. SCHOLAR 12
    2.2. WHY 14
    2.3. WEST 16
    2.4. The WUMPUS ADVISOR 19
    2.5. TURTLE 20
    2.6. MENO 23
    2.7. The MACSYMA ADVISOR 26
    2.8. DEBUGGY 29
    2.9. Repair Theory 31
    2.10. SOPHIE 35
    2.11. STEAMER 38
3. Conclusion 40
Acknowledgments 41

List of Figures

Figure 1-1:	Components of an ITS	3
Figure 1-2:	Learner control in computer-based teaching programs	7
Figure 1-3:	Brown's model of a computer coach	10
Figure 2-1:	Excerpt from a dialogue with SCHOLAR	13
Figure 2-2:	Excerpt from a dialogue with WHY	15
Figure 2-3:	Example of coaching by WEST	17
Figure 2-4:	Knowledge structures used by the WUMPUS Advisor	21
Figure 2-5:	Excerpt from a dialogue with TURTLE	22
Figure 2-6:	Program interpreted by MENO	24
Figure 2-7:	Excerpt from a dialogue with MENO	25
Figure 2-8:	Annotation of bugs in MENO's parse of a program {from (Soloway, et al., 1982)}	27
Figure 2-9:	Excerpt from a dialogue with the MACSYMA Advisor	28
Figure 2-10:	Reconstructed plan generated by the MACSYMA Advisor {from (Genesereth, 1982)}	30
Figure 2-11:	Example of a subtraction bug represented by a procedural net	32
Figure 2-12:	Analysis of a bug provided by Repair Theory	34
Figure 2-13:	Excerpt from a dialogue with SOPHIE {annotations added by John Seely Brown}	36
Figure 2-14:	Schematic from STEAMER with steps in operational procedure	39

Abstract

This survey of Intelligent Tutoring Systems is based on a tutorial originally presented by John Seely Brown, Richard R. Burton (Xerox-Parc, USA) and William J. Clancey at the National Conference on AI (AAAI) in Austin, Texas in August, 1984. The survey describes the components of tutoring systems, different teaching scenarios, and their relation to a theory of instruction. The underlying pedagogical approach is to make latent knowledge manifest, which the research accomplishes by different forms of qualitative modeling: simulating physical processes; simulating expert problem-solving, including strategies for monitoring and controlling problem solving (metacognition); modeling the plans behind procedural behavior; and forcing articulation of model inconsistencies through the Socratic method of instruction. Proceeding chronologically, examples of intelligent tutoring systems are described in terms of their internal knowledge representations and the evolving pedagogical theory. Although these programs are generally only research projects, examples of what they can do make abundantly clear the long-term scientific and software-engineering advantages of the new modeling methodology.

1. Introduction

What are intelligent tutoring systems? Why is it necessary to call them "intelligent"? Shouldn't every tutoring system be intelligent? This name in part reflects the history of the research (Sleeman and Brown, 1982, Wenger, 1986). The people who began this work—in particular John Seely Brown, Alan Collins, and Ira Goldstein—wanted to contrast their work with traditional, computer-aided instruction, so they called their programs, based on Artificial Intelligence programming techniques, Intelligent CAI (ICAI) programs. The name Intelligent Tutoring System (ITS) means the same thing.

Perhaps the best reason for attributing intelligence to these programs is their ability to solve the same problems that they present to students. This capability greatly enhances student modeling and explanation. It provides an efficient foundation for explaining all of the details of solving a problem, not just what a teacher decided ahead of time might be useful. It also allows us to build a student modelling program that can solve problems in alternative ways. Obviously, to do this the modelling program first has to be able to solve the problem in at least one way.

Explaining how a problem is solved is by no means easy, and the idea of what constitutes an explanation has changed very much in the last ten years. In MYCIN, just saying what rules were used and printing them was an innovative accomplishment. We now call this an *audit-trail explanation*, similar to an inspectable record of financial transactions. The audit-trail explanation program can tell you every step, but doesn't necessarily know the rationale behind the steps. There is a difference between saying what happened and explaining why it was the right thing to do.

Another major characteristic of an ITS is the degree of individualized instruction it provides. Early CAI programs were contrasted with classroom teaching in terms of the individualized instruction they allow. An ITS provides improved individualized instruction by building a

distinct model of what he student knows, as an interpretation of how he behaves (Clancey, 1986). Thus, an ITS relates instruction to an understanding of the individual student's goals and beliefs.

Earlier CAI programs are sometimes described as branching programs. At each point, the program makes an evaluation as to whether the student's answer at some place in solving a problem is correct or wrong. The teacher builds in the program: If the student gives answer A go to this section, if the student gives answer B, go to this section. The program can only recognize these built-in answers. No coherent model of patterns in the student's behavior is recorded. Of course, it is possible using conventional programming to build student models. For example, you could keep a history of all of the branches that a program made and have each new decision about where to branch based on the history of what branches have occurred before. This could be very complex because you would have to anticipate all of the possible histories. The methodology of artificial intelligence provides an easier way.

Intelligent tutoring systems dynamically analyze the solution history and use principles to decide what to do next, rather than requiring situations to be anticipated by the author of the program. Also, the person who writes the traditional CAI program is not expressing his strategies of why the branching is occurring at any point. At each time, when he designs a new step in his program, he may be redundantly using the same ideas of how to teach. In intelligent tutoring systems we want to extract these principles so they can be applied automatically, as well as expressing them explicitly, so we know what they are.

A final point is that we shouldn't think of intelligent tutoring systems as being one kind of program with one kind of interaction with the student. There are really many different types of programs that have different capabilities. Some are completely reactive or passive in their behavior, they wait for the student to do something and then respond. Others, like GUIDON, make an attempt to present new information in what is called *opportunistic tutoring* (Clancey, 1982). There is also a distinction between "coaching" and "tutoring." This was a point that Goldstein made in his early work (Goldstein, 1977). A coach is of course someone who watches and doesn't constantly interfere, but stays on the sidelines and lets you play the game. Then, perhaps when you ask for help or in a crucial moment of play, the coach interrupts and states an important lesson. This is a non-intrusive pedagogical strategy, based on the idea that people should just solve the problem and act on their own. If you interrupt, you won't be giving them the opportunity to develop skills to monitor their own problem solving, such as the capability to detect and back out of false starts.

1.1. Components of an ITS

Figure 1-1 is a diagram that I adapted from (Goldstein, 1978). I will go over it in some detail because these components occur in most intelligent tutoring systems.

The central idea is that there is a knowledge base, some formal model, of how the problem is to be solved, which is in the kernel of the program. For example, suppose that the student's problem is to diagnose a patient, to see if the patient has a disease and to prescribe therapy. The problem might be, as we'll see in some other examples, a game that the student is playing.

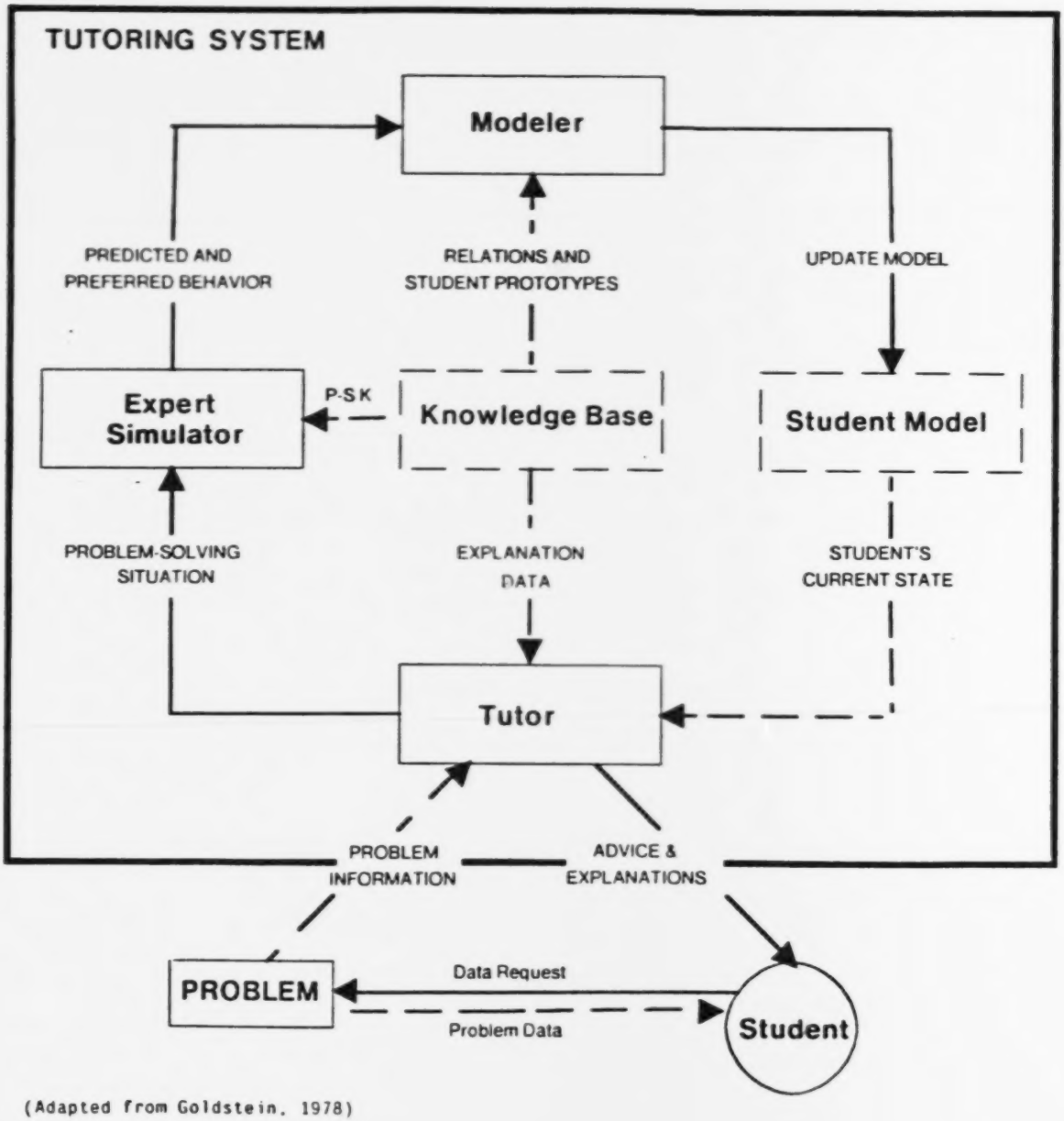


Figure 1-1: Components of an ITS

It might be a child's game, intended to teach mathematics. As the student is solving this problem, he might be receiving new information, and he's making certain solution steps that we'll call "moves" in general.

Suppose now that the student requests some patient data. The tutor is watching and passes on to a problem-solving simulator program (often called an *expert system*) all of the information that the student has received about the problem. If this were a game, such as a board game like chess or checkers, the tutor would tell the expert where all of the pieces are, the current configuration. The expert simulation program examines its knowledge base. This could include procedures, facts, plus maybe the rules of the game. The expert then generates the preferred behavior of what to do next, possibly as a set of plausible alternatives. For medical diagnosis, this would be a set of good questions to ask about the patient, and maybe an indication of the expert's preferred next question. In a game, this would be the next move to make.

This idealized information is given to the modelling program, which combines it with some additional information, perhaps about different types of students and what students know depending on their background, maybe very complex patterns. (I'm describing an idealized architecture which is more complicated than any existing ITS.) In addition to these prototypes or patterns of different types of students and student behavior, there may be descriptions of typical misconceptions or incorrect knowledge. The modeling program may also be given knowledge about prerequisite ordering of facts and procedures—what does it make sense to know given what you already know, what would it be difficult to understand given what you know? (Wescourt, et al., 1977, Goldstein, 1982)

The modelling program combines these patterns with the student's and expert's recent moves and rationalization for them, and updates the model of the student's goals, what he believes about the current problem, and what he knows in general. Thus, the core of the student model, the result of relating student behavior to the expert rationalization, is a description of what the student knows, expressed as a subset of the expert knowledge base, with degrees of belief attached to each item. For example, GUIDON's model states what the student is believed to know about the current problem, his goal structure in requesting new data, and for each rule in MYCIN's knowledge base a certainty measurement indicating whether the student knows the rule. This *differential* or *overlay* model then is fed back to the tutor, which has to decide, given principles of teaching, what to say next. Should it make an interruption and give some advice, or just let the student continue to solve the problem on his own?

Note that the separate boxes in the figure correspond to separate parts of the tutoring system. This is our programming methodology: Rather than writing one program that arbitrarily combines the knowledge, we abstract the components. This enables the expert simulator, modeling, and tutoring modules to operate upon different knowledge bases, as in GUIDON.

Are there any questions?

Question: "What is saved between sessions?" There is also an cumulative record from one problem to the next of what the student knows. Maybe some future tutoring program will save certain things that it couldn't understand about a student's behavior that, later on, it would be

able to disambiguate. In fact, this confirms fairly well to the Schank model of memory that the exceptions are saved, and they help you generalize concepts later on (Schank, 1981). Part of the problem of course is how much are we going to save; there is a space problem. This is why we need a combination of abstraction and good indexing.

Question: "Can you say more about Schank's method for remembering exceptions?" The basic idea was called MOPS (memory organization packets). Janet Kolodner did the most relevant follow-on project in her dissertation research, involving memory organization for efficient retrieval and storage (Kolodner, 1983). She models how knowledge, as a memory of many specific facts, is organized. The program is thus able to answer questions about what happened in the past and what typically happens. The idea is that generalizations form a hierarchy of concepts.

The name of Kolodner's program was CYRUS. This was a model of Cyrus Vance who was the U.S. Secretary of State at the time. She would feed the program stories of the travels of Vance and what he did on his various trips. Then you could ask the question: "When Cyrus goes to Egypt, will he visit the pyramids?" The program knows that this is a sightseeing event and looks at its memory to see where there were other sightseeing events. Is it common? Are there any times when Cyrus made a visit to a country but didn't do any sightseeing? The program would then say: "Well, yes, Cyrus will probably visit the pyramids, because every time he has been in the Middle East, he has gone sightseeing." But if you asked about Asia, Cyrus might be uninterested in sightseeing in Asia, and the memory of exceptions would indicate that visits to Asia involve no sightseeing.

The basic idea is that a discrimination net organizes and filters events in memory. The program will notice that, up to a certain point, there have been several similar events, and an exception, a different specific event, is saved. If there are two exceptions, the program looks for similarities. The network, as a pattern of generalizations, provides an efficient way of saving exceptions: An exception is defined with respect to some pattern. This is related to other discrimination models of memory, such as (Carbonell and Collins, 1973).

I found this model to be very stimulating for thinking about teaching and learning. The most intriguing thing is that this is not how knowledge bases are typically organized. In fact, Janet Kolodner is now building an expert system using her model of memory and having the program record and consider exceptions as times when its heuristic generalizations might not apply. If the program has a record of all of the cases it has solved, and if generalizations constitute its rules for classifying each new experience, the program might notice that a new case is more similar to the exceptions than to the generalizations. It will now have an ability to say that it should be careful, that this case is something different than what it can understand, and maybe it should make an attempt to generalize the exception. Of course, it's a very difficult problem to start extracting medical knowledge in this way and it requires knowledge of the mechanisms of disease; Kolodner hasn't considered causal representations. She is working in an area of psychiatry with a superficial theory of causal mechanisms, and this makes the simple classification approach tractable.

1.2. Teaching Scenarios

In Figure 1-2 John Seely Brown makes the point that teaching programs differ along a spectrum of learner control. Who is in charge of the interaction? In the case of a traditional program, what we call frame-based CAI, the branching type mentioned earlier, the program is constantly deciding what to do next. Again, this is a generalization and by no means describes every non-AI system. On the opposite side, a good example is the LOGO work of Papert and DiSessa at M.I.T., inspired by the Piaget exploratory school of discovery learning (Papert, 1980). The idea is to give the student a good environment to explore in which he can use his own innate curiosity, combining things in new ways to learn the underlying principles. If the environment is very rich, as in LOGO, with the concepts of recursion, iteration, modularity, hierarchy—which can be discovered as part of designing programs—the student may learn very general principles that can be of value to him.

When we have extremes there are often disadvantages on each side. For example, if we don't allow the student the ability to move around as he wants, he cannot use his curiosity to explore new areas. The other extreme is that, if there is no initiative by a coach or the program, the student may get stuck or unable to progress. This may also be viewed as an inefficient use of the student's time. If someone would just say; "Here is what I expect you to learn," then the student can say, "I understand that already" or, "Explain to me how this is different from something else." We can efficiently bring the student's conception of the world closer to what we think he should understand.

As you might imagine with these two alternatives, we want a combination. We want to give the student an ability to explore, but we also want some kind of an active agent, some kind of a coach perhaps that watches the student and guides him, redirecting him at various times. John Brown points out here that in some ways this is the most difficult system to build because it requires the most knowledge. If the program is always in control, we needn't understand what the student is saying to us when he wants to do something different, and we needn't do something intelligent when he asks us for something different. In the case of a program which is not attempting to present something in some logical order, there are no large and complex student modelling, tutoring, system and explanation programs to build. We just design a good language, like LOGO, and a good set of problems. It's an interesting fact about LOGO that you observe the teacher going around and helping the students, or the students helping each other. Pedagogical control can be subtle to detect and describe.

Different forms of computer-based learning can be contrasted in another way. Early programs individualized instruction by *drill and practice*. I give you a problem and you solve it; I give you another problem and you solve it; and I keep selecting problems so that they get more difficult or they follow some logical sequence. This is a good idea, and it is something that we will always want to do; sometimes you can select five good problems and if the student follows them, he will learn something. In the SOPHIE program John Brown and Richard Burton used a traditional CAI front-end to get the student familiar with the electronic circuit that they were going to learn how to diagnose. There can be a place for drill and practice.

Another perspective on computer-based learning might be called *intelligent machines*.

Locus of Control: System

Directed, individualistic learning

eg: Frame-based CAI

Shortcomings:

*Student can't explore things of interest

Locus of Control: Student

Open ended problem solving

eg: LOGO

Shortcomings:

*Student gets stuck
*Inefficient use of time

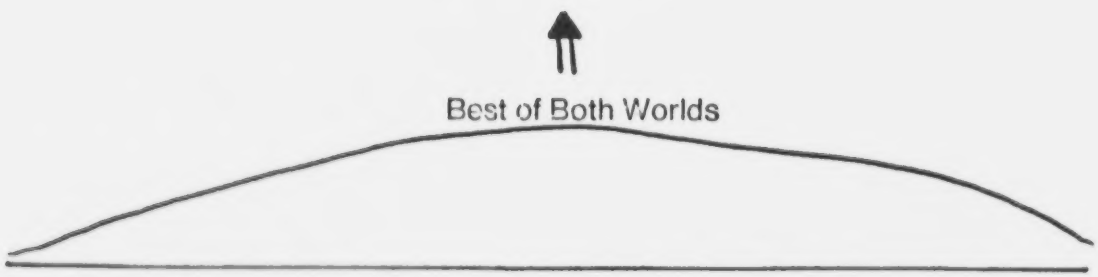


Figure 1-2: Learner control in computer-based teaching programs

GUIDON is a classic example, and there are maybe six or seven similar systems from the 1970s. These programs exploit the idea of a mixed-initiative discourse to replicate many aspects of teaching. Through the use of general teaching procedures, the program can respond flexibly to the student's initiative. Whenever the student says "help" to GUIDON, it uses its general procedures to provide help in any context. This was begun by Carbonell in the SCHOLAR program (Carbonell, 1970), which parsed natural language using a keyword parser, allowing the student to type in questions and respond.

It's an interesting historical point that from about 1979 to 1984 people stopped building these programs. The experience of building programs like GUIDON made us realize we were trying to do everything at once. We didn't have a good knowledge representation, and we didn't really understand what kind of misconceptions the students have. The idea of modelling the student became the central idea. Almost every project from the late 1970s until today has started with this one problem: How do we write a program that can understand what the student is doing? There is no need to consider strategies for teaching if we don't understand what the student is doing. It was a logical step in the research to ignore the teaching problem and emphasize the problem of understanding.

The third form of computer-based learning involves the use of *simulation*. By the late 1970s with LISP workstations and fancy graphics, it became possible to design new kinds of interaction in a teaching system. A good example is the STEAMER project. Here the idea is to use the program to give the student a realistic situation. STEAMER uses graphics and color to show dials and process change which is very similar conceptually to what happens in the world.

These three types of approaches—drill and practice, intelligent instructor, and simulation—can be combined, but they are useful extremes to consider.

1.3. Learning scenarios for an intelligent tutor

Focusing now on the intelligent tutoring form of computer-based learning, we can further categorize learning scenarios by how the knowledge in the program is used. Here we have different pedagogical philosophies.

First, we have the *Socratic* approach, in which the teacher keeps asking questions and giving new cases to force the student to realize gaps and inconsistencies in his understanding. In GUIDON, using this pedagogical approach we wouldn't spend two hours talking about one patient. Instead we would contrast cases, asking hypothetical questions and leading the student to see similarities and differences. This is the idea in the WHY program (Collins and Stevens, 1980). For problem solving that is strongly based on precedent and particular experiences, such as law and medicine, this is obviously a good way to proceed.

In contrast, a *reactive environment* is used in SOPHIE. The student solves a problem, but the program does not interrupt. It reacts to the student and gives feedback. In *learning by doing*, the apprenticeship alternative, the teacher actively contributes. The teacher watches to see whether the student is making progress and checks his understanding. After classroom learning, medical students follow the physicians around the hospital and help them solve problems. The

student takes a patient's history, and then the physician will ask, "What do you think the problem is?" He will quiz the student to test his understanding and encourage him to consider alternatives. This is essential the approach in GUIDON.

Learning while doing differs in a subtle way. Consider a job performance aid, a tutor built into the machinery of the normal working environment. An example would be a copying machine that could watch what you're doing when you make copies. Copying machines are complicated. If you want to make reverse double-sided copies with collating, there might be more than one way to solve the problem. If we build a coach into the copying machine, the coach would observe what you do. Perhaps you never use a certain combination of options. The tutor might know that people who know what they are doing use these options. Unless there is something special about your task, you might not know how to use the machine effectively, and the program will interrupt to offer assistance. Thus, a job performance aid would help you learn while you're in your normal course of business.

These examples illustrate how the research is proceeding: We are still discovering different ways in which instruction occurs and devising new opportunities for applying computers. Brown summarizes what we have learned by the aphorism that "teaching involves using knowledge, rather than just presenting it." Thus, in GUIDON we give the student a problem to solve, rather than converting the knowledge base into a multiple-choice exam. In some sense, knowledge is inherently active. What it enables you to do is important, and this is what "learning while (or by) doing" seeks to exploit.

1.4. A naive theory of coaching

In Figure 1-3, Brown points out that our theory of coaching involves a computer-based coach with many of different kinds of knowledge. He indicates the lesson that we learned in GUIDON, that it is more than an expert system, to say the very least. Yes, the tutor has expert problem-solving strategies, but it needs knowledge about explanation, how to model the student and tutoring or kibitzing strategies (an interruptive strategy for probing the student's understanding and presenting information). From all of this, we construct the model of the user, a differential model, in which difference in behavior is explained in terms of a difference in knowledge.

One thing we haven't considered here is *metacognitive knowledge*, such as knowledge about the mind, memory, and learning. Schank's model of memory organization is one example: It describes how new experience is stored and generalized. When a problem solver uses such knowledge to improve his problem solving or learning, we call such reasoning metacognition (Collins and Brown, ress). Researchers take for granted that we also need a strong theory of how people learn in order to design an ideal tutoring program. Our study of misconceptions and student models in general is just the first step in this direction. Goldstein drew a box in his early diagrams that he called the *learning model*, corresponding to this idea. Nothing in our programs corresponds to it. However, Anderson has recently incorporated a learning theory in the design of his tutors for Lisp programming and geometry (Anderson, et al., 1984a).

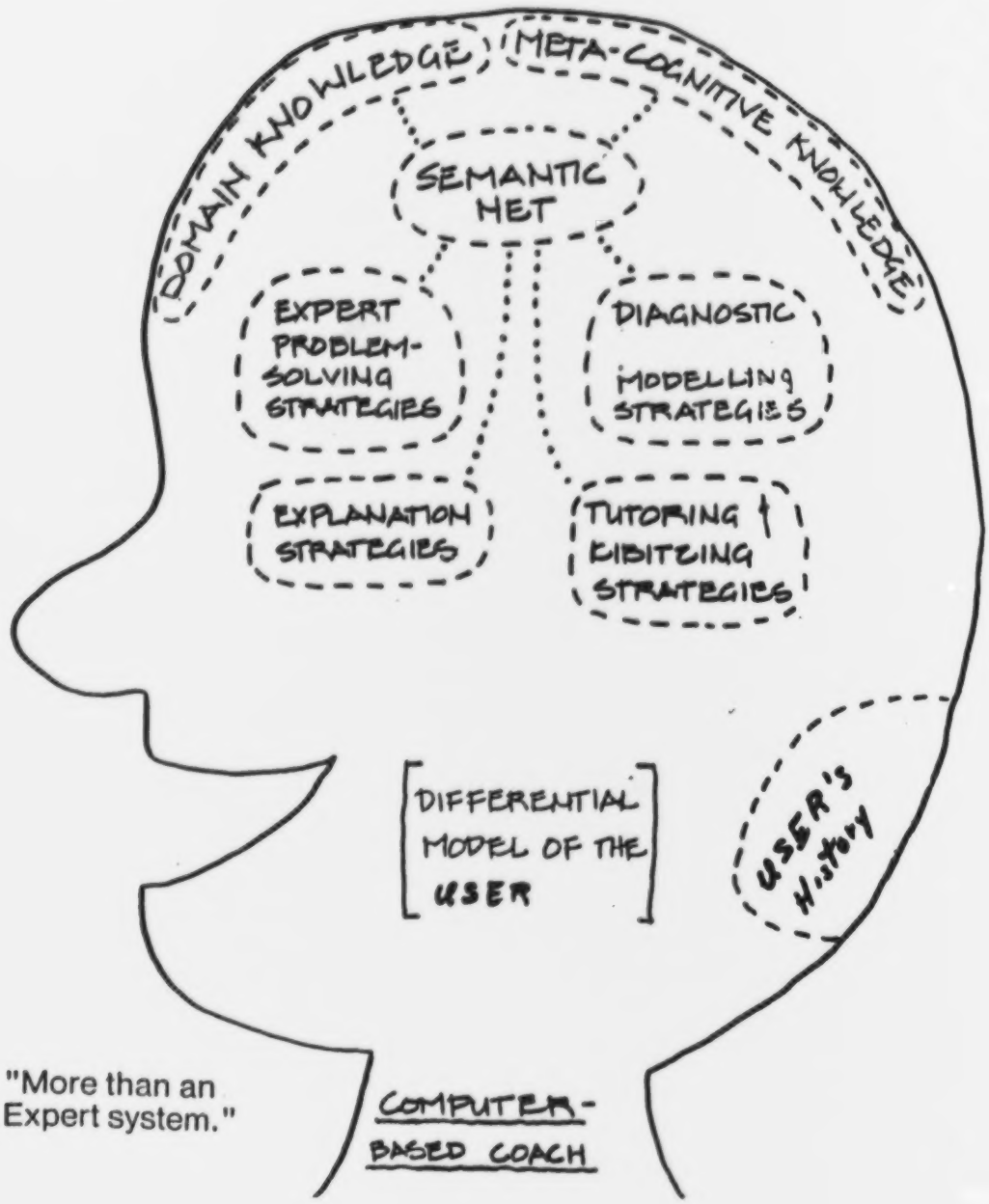


Figure 1-3: Brown's model of a computer coach

1.5. Making latent knowledge manifest

An important theme behind ITS research is the attempt to make *latent knowledge* manifest. Latent knowledge hasn't been expressed or stated, but is implicit in the behavior of a person or some physical process. By manifest, we mean articulating it in words or making it visible in pictures. A simulation shows this in a simple way. We can simulate how a device works by using graphics and show the student step by step the mechanism and the interactions. The design and the process might be very hard to see if you look at the real device; graphically it can be simplified and idealized. This is the central idea in STEAMER. Another example is modelling the knowledge of an expert; we are taking the expert's knowledge and making it manifest. An expert might not be able to state what he does, but we look for patterns in his problem solving. We say, "You have a procedure that you use. There is something you do that is regular." We make the procedure explicit.

John Brown also speaks of *reifying*—to make concrete or to make visible (Brown, 1983). For example, we can reify the space of possible ways of solving a problem. One strategy we might use for teaching is to show a student a graphic history of his various steps and compare alternative ways of solving the problem. Here is a simple example: We show the student a tree of rules and goals and show the student that if he had done things in a different order, it would not have been necessary to do as much work. This is one idea that we are pursuing in NEOMYCIN. Specifically, if the student asked a general question first, let's say about infection, rather than considering specific infections, he might save ten questions about specific infections by determining categorically that there is no infection. One way to show that is to lay out a kind of visible history and say, "You went all of the way down here in all of these details, but if you had done it this way, you could have pruned the search earlier." This is what John Brown means by reifying the problem space.

Another example of latent knowledge is the idea of a *plan* behind a procedure. Several examples occur in the programming tutor. There is the superficial behavior: What is it that the expert does? Behind that is a plan: Why is that a good procedure? To give the example of medicine again, I might ask, "Does the patient have a fever, has he lost weight, has he had a trauma, has he fallen?" This is my superficial behavior, what you see me doing when I solve the problem. More abstractly described, my procedure is to reason categorically. I'm asking very general high-level questions before going down any path. We can now state the procedure separately, as an abstraction of the original questions. Obviously, the idea of reasoning categorically has nothing specifically to do with medicine. We could also ask why it makes sense to reason categorically. If the search space is organized hierarchically, it can be more efficient to work from the top and work down rather than to work from the bottom, because you can eliminate possible paths. We call this description the plan behind the procedure.

Finally, the Socratic method is another way of making latent knowledge manifest. Asking the student questions makes him realize that there are things that are incomplete or contradictory in his knowledge. The Socratic teacher chooses examples that will force the students to realize a paradox in his behavior and in his understanding, and leads him to express a question that will resolve that misunderstanding.

In building a teaching system, we're making different kinds of knowledge explicit: the

domain knowledge, the method for explaining and the tutoring rules about interrupting and making presentations. We also make meta-cognitive knowledge explicit, knowledge about learning. To "teach" all of this to a computer involves theorizing, not just writing down what we all know for granted, but an abstraction process.

In building an expert system to be used for teaching, we learned that it is advantageous to work with good teachers. A good teacher has theories of knowledge organization and inference and learning. That is, he has knowledge about his reasoning, metacognitive knowledge, which he can impart to students to help them learn.

Similarly, because of our interest in making latent knowledge manifest, not every expert system makes a good tutoring system. In fact, we might ask whether any expert system makes a good tutoring system today. We need certain kinds of knowledge to be explicit and separate, which is the point of NEOMYCIN (Clancey and Letsinger, 1984), in which the diagnosis procedure is separated from the facts about diseases. We may need alternative ways of solving problems, including the ability to understand misconceptions. We may need to justify the knowledge base to help the student understand why it's correct. In particular, to understand misconceptions, we need to understand why the preferred rules are correct.

2. Survey of Specific Systems

2.1. SCHOLAR

The SCHOLAR program is the first attempt to use an AI representation as the basis of a teaching program (Carbonell, 1970). Carbonell saw the possibility of stating the teaching knowledge separately from the domain knowledge which was to be taught. So we have a semantic network for stating geographical facts, and we have a program for carrying out the interaction with the student. One of the later developments of SCHOLAR, work done by Collins, enabled answering questions that were not stored in the knowledge base. The program performs what is called "plausible reasoning" to infer what might be true, given certain assumptions about the world (Collins, 1978). It is a very interesting development, because people now understand this form of metacognition to be an important part of expertise. SCHOLAR is the first attempt to use AI for building a general system, allowing multiple use of the same knowledge, rather than encoding it as part of the teaching program.

Figure 2-1 shows part of an interaction with SCHOLAR, illustrating a mixed-initiative form that Carbonell and Collins sought to develop. The program asks a question, "What is the language of Argentina?" The student answers, "Probably Spanish." The program says, "Very Good," and then asks another question. A transition-network parser is used here.

You also see the student taking initiative, rather than answering the question that the program has asked. This clearly inspired Carbonell a great deal. In comparison with earlier AI programs, SCHOLAR allows the student to do whatever he wants at any point. The dialogue also illustrates retrieval of information from the semantic network. There is a hierarchy of countries and geographical areas, so it knows that Peru is located in South

What is the language of Argentina?

** Probably Spanish

Very good. Approximately what is the area of Argentina?

** Tell me something about Peru

Peru is a country.
It is located in South America.
The capital of peru is Lima.
Now answer the question you didn't answer before.

** The area of Argentina is 1,100,000 square miles

Approximately true, but a more exact value is 1,200,000.
The eastern bordering country of Peru is ---.

** Catholicism

Wrong. Catholicism is not a country but a religion.
The correct answer is: Brazil.
Where is Peru?

** Is Asuncion hot?

Since Asuncion is part of Paraguay, yes.

** What is the most important difference between Peru and Paraguay?

The primary difference is: The population of Peru is larger by a factor of 7.8.

Figure 2-1: Excerpt from a dialogue with SCHOLAR

America. It has superconcepts and instantiations of them. Carbonell also used a clever kind of response, "Now answer the question you didn't answer before." This reminds me of how people were so excited about automatic spelling correction in early Lisp programs, reminiscent of the joke about the dancing dog—it is superficial, but intriguing that it can be done at all. Carbonell also used a timer to see how long the student takes to solve the problem. The program interrupts after a while, and says: "You are taking a long time."

When the student asks, "Is Asuncion hot?", the program says, "Since it is part of Paraguay, yes." This is an example of a fact that is not stored explicitly. The program knows that if a city is located inside of a country, and country has a climatic property like temperature or rainfall, then we can infer plausibly (with uncertainty) that the city has the same property. This does not always work, and it is not too hard to think of counterexamples where there is a country with coastal and mountain cities.

More complex examples are given by Collins. The student asks whether a particular country engages in a certain type of manufacturing or trade. The program has associated with it a number indicating how important each fact is, in this case importance of various trade facts related to each country. Let's say the question was, "Does Brazil have oil wells?" The program would look at other countries and see that oil wells were mentioned, but not for Brazil, and that these wells were very highly ranked. So if a country has oil wells, it was rated at number 9, out of 10. So the program could infer that if Brazil had oil wells, and it were important for Brazil, it would be rated high, and would therefore be in the knowledge base. Assuming that you know everything about the world to a certain point, a certain degree of importance, the lack of knowledge allows you to make an assumption about what is true. This is another form of meta-cognitive ability. We can say with some degree of belief that certain things are true or false, even if we have not learned them explicitly. We can infer them from general patterns of what we know. The idea was developed further to explain how we can answer so quickly that we don't know certain things. Collins also did some psychological studies with Quillian, measuring how long it takes people to answer certain questions, and used the results to support a model of memory.

2.2. WHY

WHY is the follow-on work to SCHOLAR, developed by Collins and Stevens after Carbonell passed away (Collins and Stevens, 1980). They changed the domain from purely factual reasoning of geography to casual reasoning of meteorology, opening up many new problems.

Figure 2-2 shows the program generating certain questions for the student: "Do you think the Amazon jungle has heavy or light rainfall?" Collins has made explicit the teaching rules for generating these questions. They are stated abstractly, referring to predictions, particular cases, prior causes, general rules, insufficient causes. This kind of terminology is a major advance. It inspired me in GUIDON to express teaching knowledge abstractly, separate from the medical knowledge.

Collins attempted to mimic the Socratic dialogues that we find in the writings of PLATO. He also analyzed teachers teaching law by the case method and looked at physicians teaching

Ask for a prediction about a particular case.

Do you think the Amazon jungle has heavy or light rainfall?
** Heavy rainfall

Ask for prior causes

Why does the Amazon jungle have heavy rainfall?
** Because it has mountains nearby.

Form a general rule for an insufficient cause.

Do you think that any place with mountains has heavy rainfall?
** Yes

Pick a counter example for an insufficient cause and
Ask for prior causes.

Southern California has mountains.
Why doesn't Southern California have heavy rainfall?

Figure 2-2: Excerpt from a dialogue with WHY

medicine. Notice the systematic logic behind these questions: "Why does the Amazon jungle have heavy rainfall?" The student has just said that he thinks it has heavy rainfall. The student replies, "Because it has mountains nearby." This suggests that the student has a rule, "if there are mountains, then there is heavy rainfall." The program can look in the knowledge base to see if it is always true that mountains are correlated with heavy rainfall. The program observes that's an insufficient cause in its model of rainfall. So it asks, "Do you think any place with mountains has heavy rainfall?" This forces the student to realize that he has stated something that is not a necessary factor. Here the student says: "Yes, of course." Now the program looks in the knowledge base and picks a counterexample. We have gone far beyond just retrieving facts, which occurs in SCHOLAR, to analyzing the student's responses to see if he understands the particular model that we want to teach. Again, the idea of formalizing the questioning methods is a major accomplishment.

Collins and Stevens built in a tutoring plan that detects the facts that the student doesn't seem to understand. A teaching strategy then selects one of these misconceptions to work with. It says something like: "Deal with misconceptions before unnecessary factors." In this way, the program maintains a plan for its dialogue.

It's not clear how well WHY could understand the student's misconceptions. In their follow-on program, Collins and Stevens enumerate possible misconceptions to be recognized. After working with many students, they found patterns in what students tended to believe. The tutor checks to see if a student has these particular misconceptions. When I said, "The program now sees that the student knows the rule," I don't know how explicitly this was modelled. In particular, it does not appear that WHY followed the student during long dialogues, characterizing model changes and consistency.

2.3. WEST

WEST is an example of a coach system. It is built on top of the game "How the West was Won". It is a child's game, a variation of a game also called "Shoots and Ladders" (Figure 2-3). There are sets of spinners or dials, and the student spins these to get three numbers, which he can combine using subtraction, addition, and multiplication, or group the numbers using parentheses. The objective of the game is to get to the end first. You have the advantage of various ladders, which are shortcuts. So you want to choose the mathematical operators that will allow you to take the shortcuts. If you land on a place where your opponent is located, that bumps the opponent back. Therefore, it might be an advantage not to maximize your own forward progress, but to land where your opponent is to make him go back.

The game was originally available on the PLATO system. PLATO is the tutoring environment that was developed by Computer Development Corporation, CDC, in the 1960s. Given that this program was intended to teach mathematics, Brown and Burton decided that it would be good to add a coach to observe and critique the student's use of the various operators.

They developed a paradigm that they call *Issues and Examples*. They enumerated in the program all of the various kinds of operator combinations that a student should be able to use,

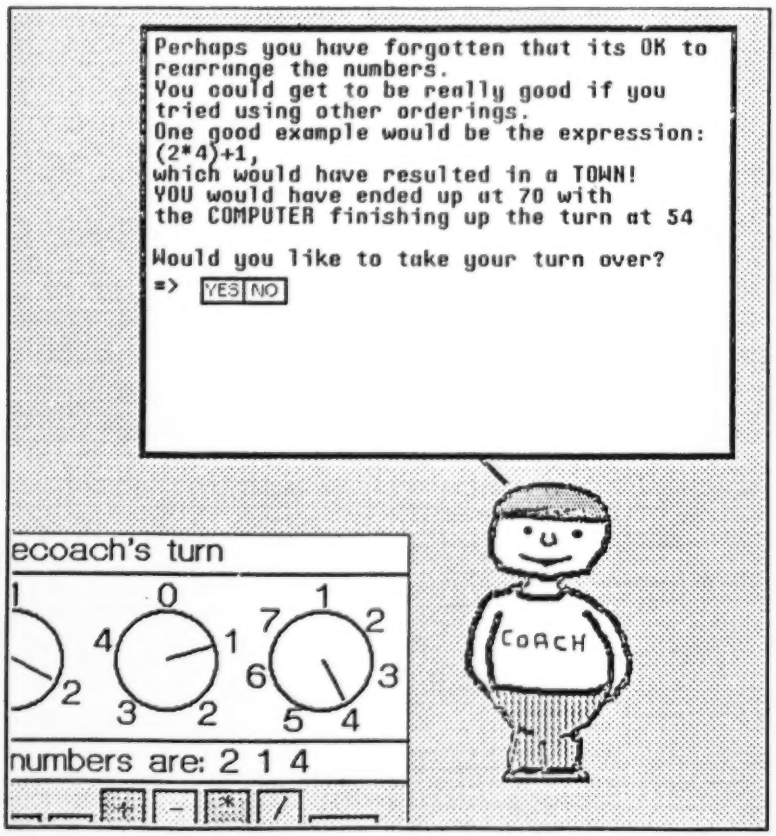

Perhaps you have forgotten that its OK to rearrange the numbers.
You could get to be really good if you tried using other orderings.
One good example would be the expression:
(2*4)+1,
which would have resulted in a TOWN!
YOU would have ended up at 70 with
the COMPUTER finishing up the turn at 54
Would you like to take your turn over?
=> YES NO
ecoach's turn
numbers are: 2 1 4
COACH

Figure 2-3: Example of coaching by WEST

called *issues*. For example, a student should be able to know that multiplication can be applied next to a parenthesis for example; this is legitimate syntax. Obviously the first thing is that the student should know is that he can use any of these operators. To give a simple example, the program watches the student as he plays the game, maybe over several sessions, and after a while notices that there is a pattern, say, the student never uses parentheses. If the student used parentheses at various times, then he might be able to play better. So, the program finds an opportunity to interrupt and says: "You never use parentheses and if you used parentheses, as in this example, you would be able to land here and you would be way down here..." The student would see that using parentheses was to his advantage. This was the idea: The interruption should be well-motivated by something that the student will understand. You shouldn't just say at some random time "You never use parentheses." Rather, pick a moment when the student will appreciate what parentheses are all about; this will help him remember. This is one of WEST's kibitzing strategies concerning when to interrupt: Pick a time when there is a good example of an issue that you want to bring to the student's attention.

The study of WEST was very complex and I strongly recommend to you the paper by Burton that appears in *Intelligent Tutoring Systems*. Their analysis of strategies is impressive. They found that there were students who were using the PLATO system with these color graphics who enjoyed the game so much, they would keep trying to go back in order to make the game last longer. This raises a problem: The student modelling program that is looking at parentheses may be missing a more general plan. So Brown and Burton invented the idea of what they called "a tear" in the model: Can you detect a time when you think you don't understand what the student is doing? For example, suppose that the student seems to be using parentheses some times, but not always, and you can't understand the pattern. One way of understanding tear is that there are constant breaks in known patterns. It raises the larger issues of how do we detect strategies, and how do we detect if a model is good or bad, and what does it mean to recognize new strategies that were not built into the program?

The issues and examples paradigm is very useful for understanding how to build a tutoring or teaching system. The idea is very simple. You build into the program concepts or rules (the issues) that you want to teach the student, in this case the use of mathematical operators. Then you build in recognizers (procedures) for detecting whether or not the student knows each issue. In GUIDON the issues are the EMYCIN rules, and the recognizers are replaced by a single, general modelling program, which attempts to determine whether the student knows the rules used by the EMYCIN expert program. In contrast, in WEST each issue has a program for recognizing it, called a *specialist*.

Today in the US there is increasing interest in applying some of these early ideas, especially programs like SCHOLAR and WEST. At San Francisco State University, and in several other universities in California, they are trying to teach high school teachers how to use this paradigm in their own areas of expertise. They are receiving computers from Hewlett Packard, new Lisp machines and networks. This is encouraging because it is something that researchers don't have the time to do—setting up projects to teach high school teachers what we did ten years ago and trying to make the ideas practical.

To summarize the research paradigm, we are stating teaching rules in an attempt to formalize principles. We've considered three examples: The WHY system illustrates the principle of constructing a counterexample for an insufficient factor. In GUIDON (in an earlier lecture) we considered part of the procedure for completing a topic; GUIDON simply states a domain rule rather than belaboring the discussion. In WEST, we considered the principle of illustrating an issue by an example that is dramatically superior to the move made by the student. I think this shows a science in a very early stage of writing down what is intuitively, in common sense terms, believed by the researchers to be reasonable.

After this, we must represent this knowledge explicitly so it can be reasoned about by the programs and can be explained. Every complex system like GUIDON will have hundreds of teaching rules. They form another knowledge base, that we want to be easy to change. The program has to be able to explain its behavior, easy to modify, and so on. Next, we must study this formalized knowledge and understand why these are good things to do. What is the underlying model of learning and of human reasoning that makes these good teaching rules? For example, if we look at fifty GUIDON rules, we might find that they are inconsistent. Maybe we want to build different versions of GUIDON that systematically test alternative ways of teaching. For example, Collins studied Socratic tutoring in many different domains.

This is a good place to mention the work of Beverly Woolf. She went back to GUIDON, studied it in some detail, and restated the tutoring discourse according to natural language conceptions of patterns in a dialogue. She essentially tried to extract the strategies that were implicit in the alternative ways of tutoring that I had compiled together in GUIDON's rules. Her program, the MENO-TUTOR, reasons in more layers of abstraction about what to do (Woolf and McDonald, 1984).

2.4. The WUMPUS ADVISOR

Goldstein and his colleagues developed a coach for the WUMPUS game, adapting it for teaching probabilistic reasoning to a student (Goldstein, 1982). In a manner similar to SCHOLAR, WEST, and GUIDON, Goldstein used production rules for stating teaching principles. Various rules produce explanations and select examples. In the game, there are many caves, bats and pits. You move through the caves to find your way out or to find some treasure. When you arrive in a particular cave, you receive information about what might be nearby. You are told that it's cold or that you can hear certain sounds. You know that pits will have cold air coming from them, and so on. As you go through this network of caves, you can collect evidence of what is nearby.

At one point the coach might interrupt and say, "Mary, it isn't necessary to take such large risks with pits. We have seen that multiple evidence is more dangerous than single evidence for bats...." Figure 2-4 shows part of WUMPUS's reasoning here. We want to teach the student the general rule preferring single dangers over multiple dangers, and this becomes instantiated or specialized for three different types of dangers. Thus, here a previous explanation about how the rule applies to bats is referred to when explaining its application to pits. Goldstein points out that the program needs to keep records of what it has explained already so it has an opportunity to draw an analogy. Notice that in this formal domain,

unlike medicine or meteorology, there is no underlying causal model, just axioms describing how the world works. This greatly simplifies the modeling problem.

2.5. TURTLE

The next three programs are concerned with teaching computer programming. We see a student's program and want to understand its design. What is the student's plan? Why is he using these steps in this way?

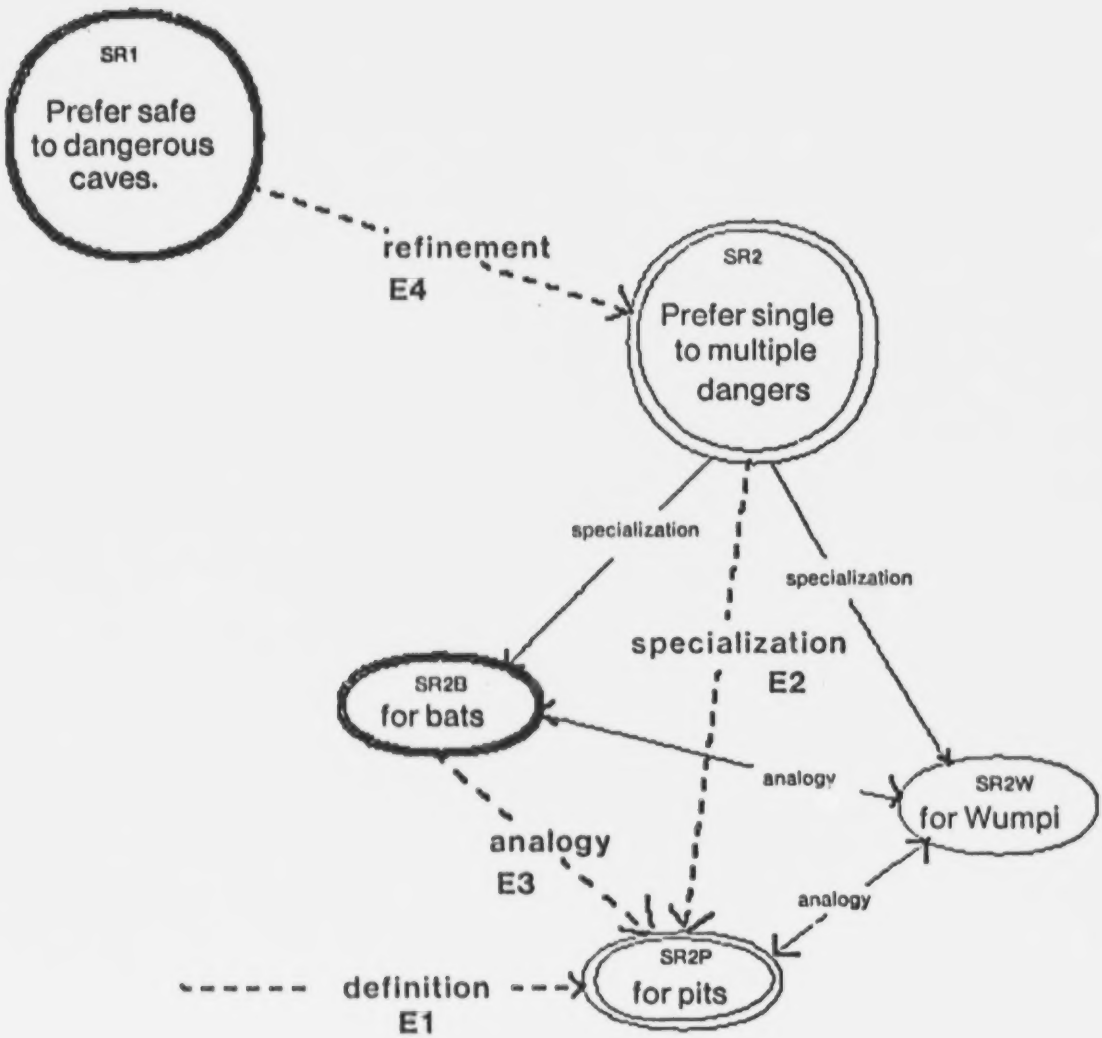
"Turtle" was one of Papert and DiSessa's constructions for teaching using LOGO (Papert, 1980). The instructions in the language are very simple, you can simply turn the turtle or you can move in a certain direction or you can have it lower its pen and draw as it moves. They actually have a mechanical device that draws on paper in the room; the student types in the program and the little robot turtle moves around. In this way the students can watch what their programs do, another example of reification. This was part of Papert's idea of an environment that would give students feedback.

The TURTLE system is extremely simple; it has the capability to only write four programs, and the solutions are canned (built into the program). The program is able to detect certain types of errors. It has prespecified hints that it prints out, which are intended to help the student (Figure 2-5).

Here is how the program proceeds. The problem is to draw a well with a roof on it. We have a box, a square and then a pole. It's like a triangle on top of a box. Miller analyzed in detail the various ways that people solve this problem. In the interaction the student says that his first step is to draw the top using another procedure. Then he wants to have the pen go 200 steps forward, draw a box, and then stop. This is his program for drawing a well with a roof. The program has to relate the terminology of the student's names for his programs to the internal names that the programmer has defined. This is a very interesting problem in programming tutors. The modelling program has to understand what the variables stand for. The program infers here that "Top" is the roof of the well and "Draw 200" is the pole and "Box" is the student's name for the base. This was done by building into the program various synonyms that people are likely to use.

Figure 2-5 shows a mapping between the student behavior and the internal model of the task. It shows in a simple way the importance of being able to recognize alternative solutions, in this case by using something as trivial as different terminology. Next, the student defines these functions for Top and Box, and the program analyzes where the pen moves around.

Next the TURTLE program points out that there is some basic step that has been left out. It says, you have provided no "Set-up" for the Well task. "Set-up" is a general programming concept. Before you call another procedure you have to get the turtle in the right position to do the next procedure. It's a kind of initialization that makes the next procedure perform correctly. If you look at the picture, the roof might have been sideways, or maybe the pole might have the wrong angle. The program analyzes what the drawing looks like and relates it to what the correct step should be. It points out that two steps were left out as part of this interface.



A dark outline signifies a rule hypothesized to be known by the player.
A double outline signifies a rule explained to the player.
A dashed link signifies that an explanation corresponding to that link has been generated.
E1 through E4 are the corresponding explanations in the text.

Figure 2-4: Knowledge structures used by the WUMPUS Advisor

Task is to draw a well with a roof.

First step? TOP
Next step? DRAW 200
Next step? BOX
Next step? QUIT
OK, WELL is defined!

Let me see if I understand your program:
The function TOP draws the ROOF of the WELL-TASK.
The code: DRAW 200 draws the POLE of the WELL-TASK.
The function BOX draws the BASE of the WELL-TASK.

Does that seem correct? YES

After the student defines functions TOP and BOX

Did you intend to interface the function TOP to the step DRAW 200 by following the path C D? YES

Here is the interface you defined between the mainsteps in the function TOP and the step DRAW 200 in WELL:

TURN 180

Analysis:

Correct code	Commentary
TURN 180	Correct interface to turn through angle BCD
MOVE 50	Missing interface to move over segment CD
TURN 90	Missing interface to turn through angle CDE

. . .

You provided no setup for the WELL-TASK.

Figure 2-5: Excerpt from a dialogue with TURTLE

This is a very simple example of conveying to the student that programming has certain plans behind it and there are certain recurrent operations in programming, in particular, performing a set-up as part of calling a subprocedure. The program has related the student's behavior to the underlying model, isolated an error, told the student what the error is, and indicated what he should do differently. Notice that there is no attempt to understand the student's error, why he do this, just what he did wrong and the correction that should be performed.

2.6. MENO

MENO (Soloway, et al., 1981) is the work of Elliot Soloway, who is now at Yale and did his thesis work at the University of Massachusetts in the late 1970s. MENO attempts to understand a student's programming bug, to articulate the misconception. It not only fixes the plan as in TURTLE, it helps the student resolve his misconception. To summarize these levels: here is the surface behavior of the program; there is the plan, which is the design; and there the underlying misconception that generates the design that generates the program.

gure 2-6 shows the input to the MENO program; it is a very difficult task to understand a gram like this. It requires knowledge of the problem that is to be solved and knowledge out alternative solutions. The purpose of this program is to compute an average. It itializes the sum and then counts how many numbers have been input. While the input is ot equal to 999 (a *sentinel*, a strange number that would not be part of the input) we add the input to the sum and increment the count. Notice that the student adds 1 to the input and then continues. This obviously takes a long time, but eventually we get to 999, exit the loop, and divide the number of times we've gone around into the sum.

The student has made a mistake of putting the READ statement outside of his loop. Why did he do that? Soloway and his collaborators have performed a very interesting analysis to get at the misconceptions.

Consider what MENO says to the student (Figure 2-7). First, like TURTLE it finds a mapping between the student's names and the internal names of the program. It says, "Positive identifier is the new value variable and count is the counter variable." There is a general plan that MENO has for doing averages. It doesn't have variable names like X, W, A, B, but it has general names that correspond to the meaning of the variable: to get the new value, the counter, the running total, etc. And so again, we're relating the student's behavior to the plan that's in the program. MENO has many plans for the problems that it solves. The program points out the bug, "You modified the new value variable by adding 1 to it whereas you should modify it by calling the READ-procedure."

MENO then goes further than TURTLE. It has a library of misconceptions, not generated by the program, but built-in by Soloway. One possibility is that the student thinks that READ is like a declaration and so, every time he mentions X, its value should be read. It is like saying that X is an integer, describing a property that should persist through the entire program.

Figure 2-8 shows the kind of analysis that is being performed. Meno builds a semantic net,

A student's program...

```
1  PROGRAM AVERAGE1(INPUT, OUTPUT);
2   VAR
3       SUM, POSIDEN, COUNT: INTEGER;
4          AVE: REAL;
5    BEGIN
6          SUM: = 0;
7          COUNT: = 0;
8          READ(POSIDEN);
9          WHILE POSIDEN<>9999 DO
10            BEGIN
11               SUM : = SUM + POSIDEN;
12               COUNT : = COUNT + 1;
13               POSIDEN : = POSIDEN + 1;
14            END;
15         AVE : = SUM / COUNT;
16         WRITELN('THE AVERAGE IS ', AVE)
17   END.
```

Figure 2-6: Program interpreted by MENO

MENO's analysis...

POSIDEN is the New Value Variable
COUNT is the Counter Variable
SUM is the Running Total Variable

You modified POSIDEN by adding POSIDEN to 1
where as...
you should modify the New Value Variable by calling the READ procedure: READ(POSIDEN).

Two misconceptions can be associated with this bug:

1. You might be thinking that the single call to the READ procedure at the top of your program is enough to define a variable which will always be read in from the terminal....

2. You might be thinking that POSIDEN is like COUNT... The computer does not know to reinterpret + 1 in the former case to be like a READ.

Figure 2-7: Excerpt from a dialogue with MENO

a parse of the program, and then relates each of the variables and the operations to an internal model of the problem. Then it associates certain bugs with various plans. The analysis is not generative; the program cannot take an arbitrary program and understand it. It has to be an "Average program" or another one in the library. Explaining bugs requires a lot of creativity. The explanation of the bug as an alternative model is often as complicated and difficult as what you are trying to teach. People probably rely a great deal on experience and receiving information from students. It's uncertain how many teachers could understand what the students are doing. We are certainly far from being able to write a program that could generate similar explanations for misconceptions, especially in domains like medicine or programming.

MENO was re-implemented by Lewis Johnson in his dissertation research as the PROUST program (Johnson and Soloway, 1984). Rather than matching bugs as specific segments of code, it constructs a complete parse of the code on different levels of abstraction. This provides a more robust, generative capability for understanding student programs and is the first step beyond the template approach of TURTLE and MENO. The misconceptions are still built-in, but the program bugs are stated abstractly, rather than as specific lines of code to be matched.

2.7. The MACSYMA ADVISOR

The MACSYMA Advisor (Genesereth, 1982) is very similar in style to the MENO system. We have the same problem of understanding a sequence of student behavior. We want to extract the user's plan. MACSYMA, developed at MIT by Joel Moses, is a mathematical manipulation system for simplifying and combining the equations. The problem is to take a user's sequence of interactions with MACSYMA and to provide help or consultation.

Figure 2-9 shows an example in which the user is surprised by an answer of zero. The user asks for help, indicating that there is a problem. The point is, when we type in commands to a program like MACSYMA, we have some kind of plan in mind, that is, a method or approach for solving a problem. In this case the user was trying to solve an equation (to determine values of X for which the formula evaluates to zero). The MACSYMA Advisor has to examine the sequence of user actions, and given the user's goal, infer his plan for using the MACSYMA program. The Advisor has a built-in library of different ways of using MACSYMA, somewhat similar to MENO's library of program patterns. The Advisor provides a remediation in terms of correct set of steps of what to do, plus an explanation of the student's behavior in terms of a misconception.

The user says, "I was trying to solve equation that was stated in D6 for X and I got 0." At this point, the program determines that, if this is the user's goal and this is what he did, he must believe that the COEFF command or operator has a certain effect. The user has a misconception. He believes that COEFF returns the coefficient of a particular formula, but it returns the coefficient of X to the first power. The program goes on to say that, "If this is what you are trying to do, it's important that the expression be expanded with respect to the variable, and you should use RATCOEFF to get the right result."

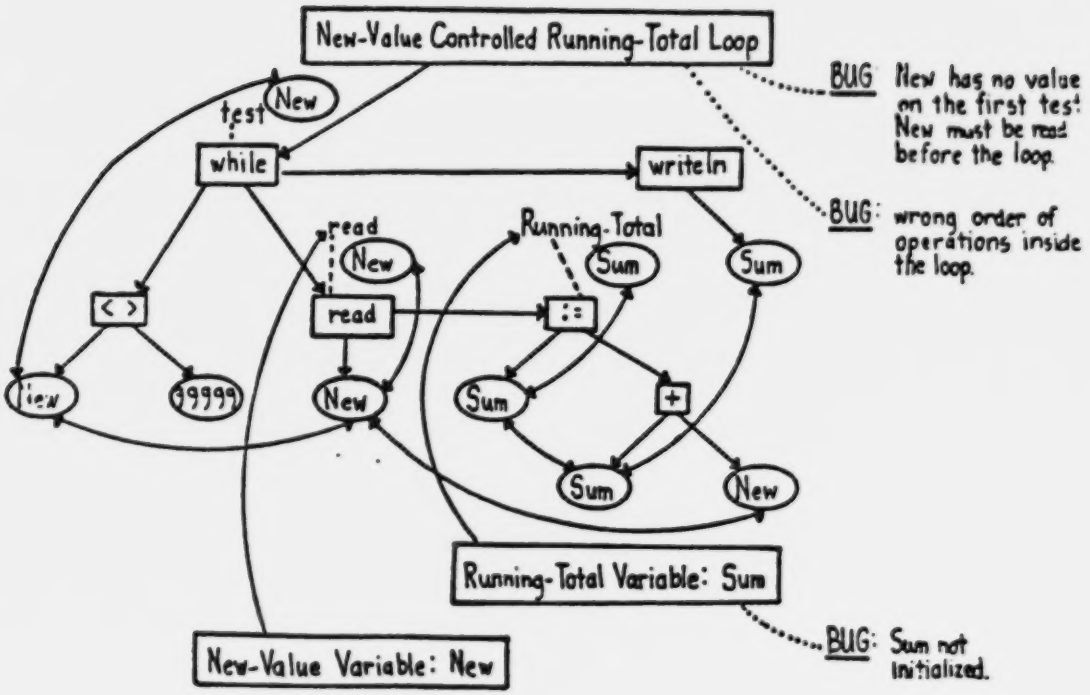


Figure 2-8: Annotation of bugs in MENO's parse of a program {from (Soloway, et al., 1982)}

```
(C7) (A:COEFF(D6, X, 2),  B:COEFF(D6, X, 1),
      C:COEFF(D6, X, 0))$

(C8) (- B + SQRT(B**2 - 4*A*C)) / (2*A);

(D8)                0
```

(C9) HELP()$

USER: I was trying to solve D6 for X and I got 0.

ADVISOR: Did you expect COEFF to return the coefficient of D6?

USER: Yes, doesn't it?

ADVISOR: COEFF(exp, var, pow) returns the correct coefficient of var^{pow} in exp only if exp is expanded with respect to var. Perhaps you should use RATCOEFF.

USER: Okay, thanks. Bye.

Figure 2-9: Excerpt from a dialogue with the MACSYMA Advisor

The important point is how the MACSYMA Advisor constructs this interpretation. Unlike the other programs we've seen, where the misconceptions were pre-enumerated, the MACSYMA advisor has the capability of generating the misconception by an analysis of the student's plan. Consider Figure 2-10. At the top, you see the goal of solving the equation for X. At the bottom appears the user's superficial behavior. In the middle appears the interpretation of how this goal generates this sequence of behavior, expressed as a hierarchical structure of subgoals and methods for accomplishing them.

To make the user's actions fit together in a coherent explanation, the program makes certain assumptions about the user's beliefs. The Advisor indicates the actual input and output for each operator and general constraints that the input and output must satisfy. Thus, the interpretation of the user's misconception, as a model of his beliefs, is stated as both specific computation results and general facts about MACSYMA's operators.

In particular, the analysis shown here is made complete by assuming that the user believes that this particular operator, COEFF, returns a particular value that has a certain input and output. This assertion about COEFF is technically incorrect, but it makes the plan coherent; the user's sequence of behavior is now understandable. Constructing such a plan is a complex search problem. The program generates multiple parses in both a bottom-up and top-down way. Having the behavior constrains possible interpretations, and knowing the goal constrains the possible operations that are performed. Different template (canned) plans are used by the program to generate and to recognize the sequence of behavior.

We see here the power of a logic formalism in which Genesereth expresses the various constraints on each of the operators and what the operators accomplish. Again, an appropriate language enables writing a program that can reason about the knowledge involved. If these operators were in LISP, the plan recognizer wouldn't be able to reason about them. Notice how far this is from the SCHOLAR system, which has only an idea of right and wrong answers. Here we generate a structural analysis of what the user believes, given his behavior. I think this is one of the best examples of student modelling. It shows us how to generate misconceptions, rather than building them into the program. Contrast this with the work of Stevens and Collins where the misconceptions about meteorology and heavy rainfall are pre-enumerated, so the program has only a fixed library of possible misconceptions. It helps of course to be working in a mathematical, closed domain. Genesereth has gone on to develop the ideas of the representation system that was used here, MRS, and to examine in more detail architectures for problem solving (Genesereth, 1983).

2.8. DEBUGGY

DEBUGGY is a program for modelling a student's knowledge of the subtraction procedure. It models incorrect procedures, to be contrasted with modelling (factual) misconceptions. DEBUGGY builds a procedure that describes how a student does subtraction, based on a set of example problems solved by the student.

Figure 2-11 shows the procedural network that is DEBUGGY's internal representation of an incorrect subtraction procedure (Brown, 1978). It shows a decomposition of the various steps.

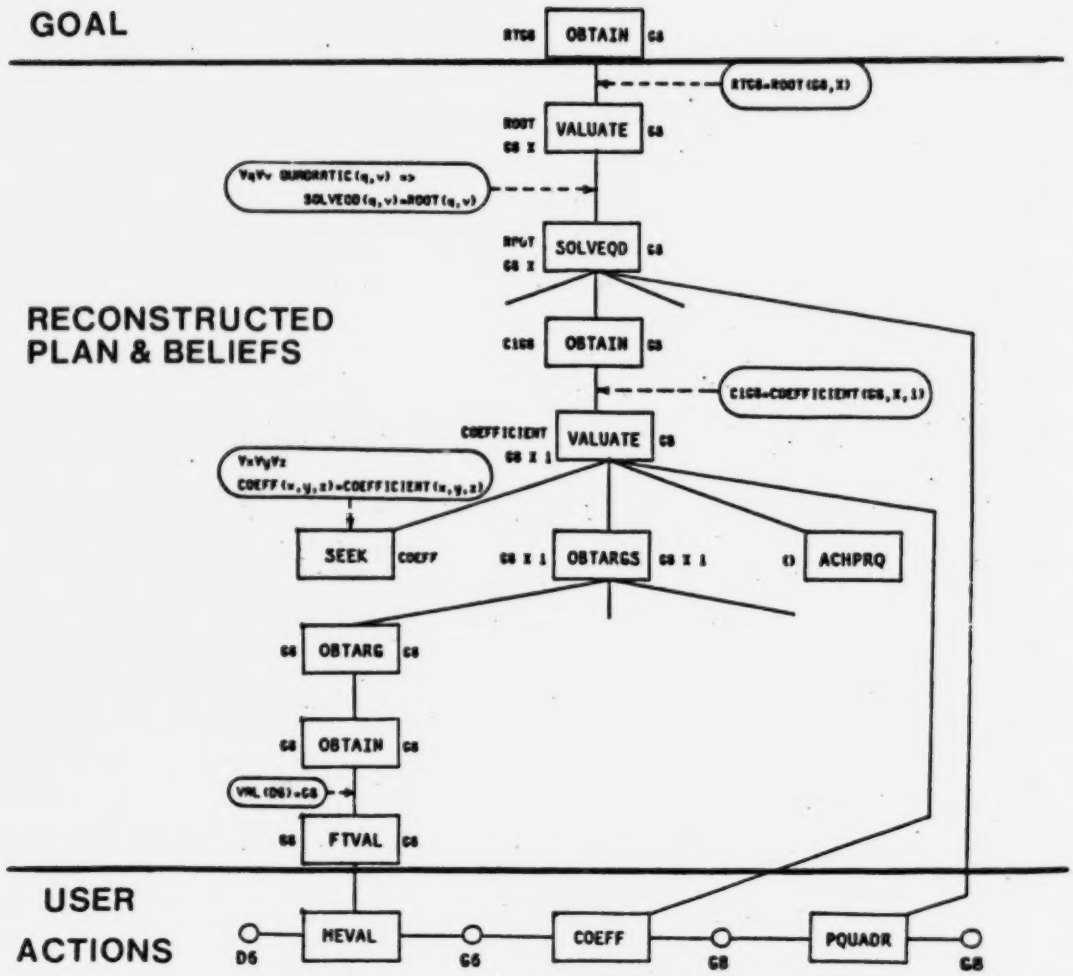


Figure 2-10: Reconstructed plan generated by the MACSYMA Advisor {from (Genesereth, 1982)}

If for example, at one point, you want to subtract a column from another, there are various alternative procedures to apply depending on the current situation. Here the subprocedure "switch digits" has replaced "borrow and subtract." Brown and Burton were inspired by Sacerdoti's approach of using a procedural network to describe a plan.

The idea of DEBUGGY is that there are systematic errors that people make and we can analyze the problems to detect these errors. You should realize that there is significant computational problem in generating this description from all of the possible alternative procedures DEBUGGY can construct. On the order of a hundred bugs were found to occur in students. There were thousands of students whose problems were examined.

2.9. Repair Theory

Recalling the distinction between describing student behavior and the rationale behind this behavior, Kurt van Lehn's project was to understand where subtraction bugs come from. We start with the surface behavior: The student can't subtract from 0 or doesn't subtract from 0. We describe the procedure, and now we ask, "Why does he do that?" Where did this bug come from? It is a model based on how people learn. In repair theory, rather than just recognizing the bugs, we try to understand the cause of the misconception. This is like taking the MACSYMA Advisor one more step. Suppose you have isolated a misconception, what is the reason for it? It's an attempt to understand learning problems. Recalling the MENO example, why does the student believe that READ is a kind of declaration. How can we explain why students have certain bugs and not others? We want a theory of where the bugs come from that allows us to generate the bugs and explain the systematic errors in more primitive terms.

The central idea in repair theory is that an incomplete correct procedure causes an impasse during problem solving, which is repaired by a general problem solver and critics. This is an interesting first-order theory which seems to have a lot of power. In one sense, this is the idea of an *overlay model* again: The students know a subset of the correct procedure. But the next part is different. Suppose a student doesn't know what to do when there is a zero on top or when it's necessary to borrow (in a subtraction problem), but they do everything else correctly. Without a complete procedure, the student gets stuck. This is called an *impasse*.

For example, if you don't know what to do when the top number is smaller, you have to do something. You're supposed to write down some number when you're doing a subtraction, you can't just leave it blank. Students know that and so they need some kind of alternative way of acting. When they reach this impasse, they repair their procedure in some way. This is the origin of the name "repair theory." vanLehn assumes that people have general problem-solving methods that they will follow when they get stuck. They don't sit there paralyzed, they do something. He wanted to study how a repaired impasse might generate bugs.

Figure 2-12 gives an example. Suppose we take the bug of not being able to borrow when the top digit is smaller than the bottom digit (not being able to subtract 6 from 3 here). This is modelled by deleting part of the procedure that allows us to behave when we reach that situation. It's a simple production rule view of subtraction. There are situations and operations you can perform. If you see a large number on top of a small number, you just

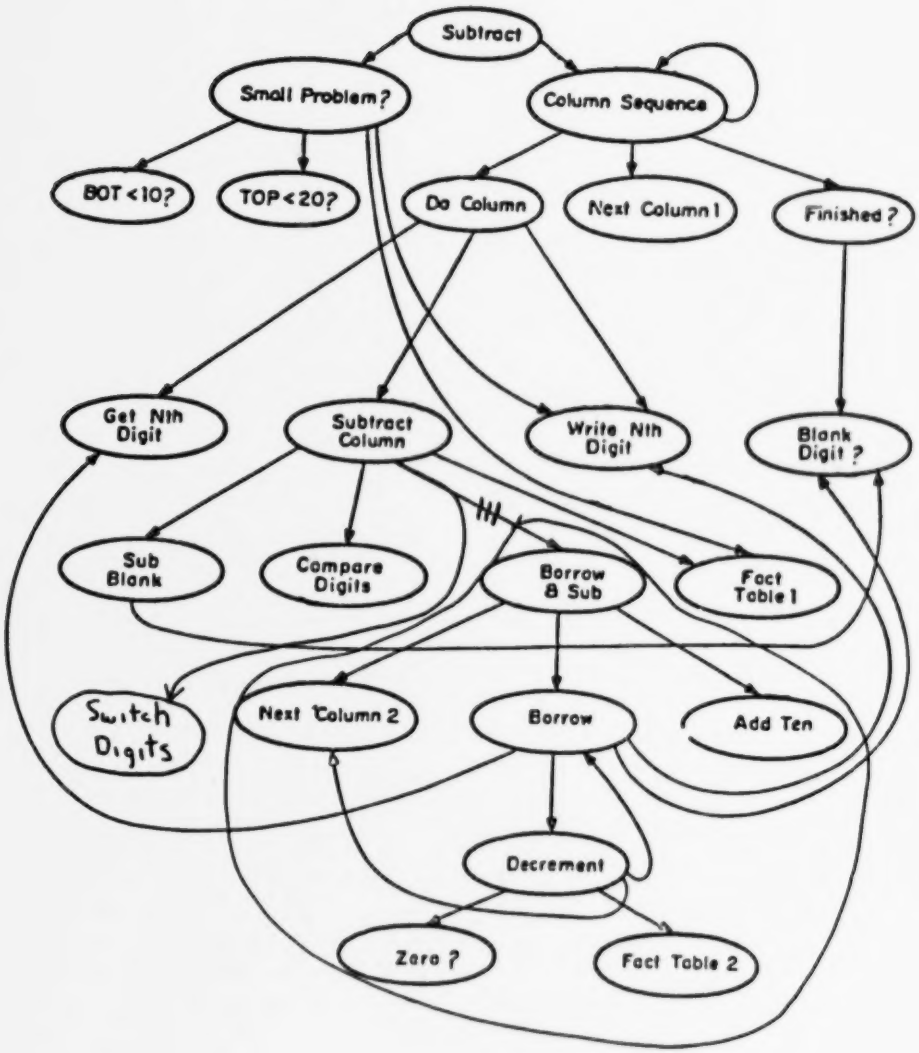


Figure 2-11: Example of a subtraction bug represented by a procedural net

look it up in your table. So if you see 8 over 2, you can look it up: 8-2 = 6.

But now you get to 3-6, and let's say that there is no table for that. You can't write down -3. It's not permitted to write down a negative number here, but you have to write something down. So it is time to borrow. But if we delete the knowledge about borrowing, there is an impasse.

vanLehn gives the general problem-solving strategies that he believes are sufficient to generate the observed bugs. First is to skip the column, write down nothing. Some students do that. Second is to quit, stop the problem right there. Don't write down any more numbers, go on to the next problem. Third, (this is very creative) just turn the numbers around and subtract. Fourth, called *dememoize*, just look up the answer in an idiosyncratic table of arithmetic facts.

These repair strategies will generate different bugs. We have the opportunity now of systematically deleting rules in the correct subtraction procedure, applying the problem-solving strategies and generating the bugs. We can go backwards from actual problems and determine how the people applied the strategies and so on. DEBUGGY discovered 89 different bugs in several thousand problem sets. Repair theory explains (generates) 21 bugs that have been observed. Also interesting, it generates 10 bugs that have not been observed but are at least plausible or possible. VanLehn uses the term *star bug* (from language theory) for a bug that we, for many reasons, believe could never occur. (This is from the idea of a sentence in a language that we would never expect anyone to utter.) Obviously we want the fewest number of star bugs; VanLehn takes this as a measure of quality of his theory that there is only one star bug. As for the ten originally unobserved bugs, he went back and found some of these in the problem sets. There are many detailed parts of this theory; it has recently been extended to explain the origin of impasses in what is called *Step Theory* (VanLehn, 1983).

Finally, *bug migration* concerns changes in bugs over time. Using Repair Theory, VanLehn and Brown were able to make certain predictions, based on the idea if the student has one bug, he might replace it by another one because his impasse says the same. He still doesn't know how to borrow, but he might apply a different problem-solving strategy. One time he might leave it blank, and the next time he reverses the numbers. Thus, the bugs are related to one another, and vanLehn and Brown found a systematic change from, say, Friday to Monday when retesting the student. This also turns out to be an interesting concept because, originally, we might have said the behavior was random because the bugs seem to be changing. But a deeper analysis shows that there is some systematic pattern. Vanlehn has written extensively about the problem of developing a principled theory (VanLehn, 1984). I've gone into some detail with his work because even though DEBUGGY and Repair Theory are not tutoring programs, they provide a good example of how AI methods are used to develop detailed models of human problem solving.

1. Delete rule of procedure:
   "borrow when top digit is too small"

2. Problem solving ends with an impasse:

```
  638
- 462
-----
   ?6     => impasse occurs in second column
```

3. Different repair methods produce different procedural bugs:

REPAIR	BUG
Skip	blank-instead-of-borrow
Quit	doesn't-borrow
Swap vertically	smaller-from-larger
Dememoize	zero-instead-of-borrow

Figure 2-12: Analysis of a bug provided by Repair Theory

2.10. SOPHIE

SOPHIE is a very large, complex program of three different versions developed by Brown and Burton from about 1974 through 1978. Most of this work was done at Bolt, Beranek and Newman. SOPHIE stands for "Sophisticated Instructor for Electronics," but it was also the name of Richard Burton's dog. SOPHIE versions 1 and 2 did not contain an expert system in the sense that we define it today. At their core is a mathematical FORTRAN program that simulates how a particular circuit behaves under different loads. The simulator is used by the tutoring program to evaluate a student's diagnostic hypotheses. SOPHIE 3 is an attempt to formulate a procedure for doing diagnosis, an expert system, that the student can watch and emulate.

Recall that SOPHIE provides a reactive environment; it does not attempt to teach anything to the student in a systematic way. The student is given a version of the circuit with a bug in it, or a *fault*, as it is called in this context. He is to determine which component is faulty. The student makes measurements and then poses a hypothesis. He can ask questions about what could possibly be wrong, given the information is received.

Consider Figure 2-13. The program starts with a fault in the circuit. The student is told various readings, such as voltage. Burton developed a "semantic grammar" for parsing the student's input. This parser consists of a set of procedures corresponding to various phrases and expressions. The parser is capable of disambiguating pronouns and filling in omitted references (anaphora). The program has a fact table in which it looks up circuit information.

The student makes a measurement, "What is the voltage between R8 and Q1?" Later he poses a hypothesis, "Is it possible that Q6 is open?" The program has to disambiguate this description, and then, using the simulator, determine if the hypothesis is consistent with the current setup of the circuit. It turns out that this is not the fault. The program points out that, if it were the fault, the measurements would have been different. To do this, it sets up the circuit again, using the student's fault and runs the FORTRAN simulation to determine how the circuit would behave given the fault that the student has stated. Using the FORTRAN simulation program in this way requires considerable intermediate knowledge structures and reasoning to set up meaningful boundary conditions for the hypothesized situation. This integration of multiple knowledge representations is a major contribution of SOPHIE research: the FORTRAN program simulates the circuit; a semantic network describes the components of the circuit; and semantic parsing procedures disambiguate student input.

The program comments that even though the student's hypothesized fault is incorrect, it is consistent with two of the values that the student received. We've seen this tight logical analysis of consistent, necessary, and sufficient factors in several programs (WHY, GUIDON, WUMPUS ADVISOR).

The student could also ask, "What are all of the things that could be wrong?" This has to be answered with respect to the current known measurements. The program has to run multiple simulations, setting up the circuit in different ways to determine all possible faults that could account for the behavior that's been observed so far. This is also a very complicated type of analysis.

Welcome to SOPHIE, a reactive learning environment

The instrument has a fault and its present control settings are:

CC = 1, CR = HIGH, LOAD = 20, VC = 1, VR = HIGH ← Front panel settings

>>What is the output?
The output voltage is 11.7 volts.

uses model of dialog for reasoning pronouns

>>What is it in a working instrument?
In a working circuit the output voltage is 19.9 volts.

>>What is the voltage between R8 and Q1?
The voltage at N1 is 47.6 volts.

>>What is the base emitter voltage of the voltage limiting transitor?
= transistor
The base emitter voltage of Q5 is -.037 millivolts.

hypothesis evaluation

>>Is it possible that Q6 is open? ←
Is the complete fault mode EB/EC/BC = OP/OK/OP what you intended?
>>Sure

constructs counter-example

When the settings were:
CC=1.0, CR = HIGH, LOAD = 20, VC = 1.0, VR = HIGH ↙
That fault conflicts with the following measurement. You observed that the output voltage was 11.7 volts. If Q6 had fault mode EB/EC/BC = OP/OK/OP it would be 23.9 volts. In a working circuit it is 19.9 volts.

That fault does not explain any of the symptoms you observed but it is consistent with the values for the base emitter voltage of Q5 and the voltage at N1.

← but also determines what is "good" with the hypothesis.

Figure 2-13: Excerpt from a dialogue with SOPHIE
{annotations added by John Seely Brown}

While SOPHIE has no teaching knowledge in it, it provides a laboratory work bench, a kind of smart tool. In any lab, we could take a faulty circuit and have the student try a diagnosis. But here using the computer with a simulation, the student can pose hypotheses and ask for alternative analyses.

We can summarize the type of inference that SOPHIE can perform as follows:

1. Hypothetical question answering: What happens when the circuit is changed in a certain way?

2. Hypothesis evaluation: Could the circuit be faulty in some way?

3. Hypothesis generation: What could be wrong, given the information known so far?

4. Redundancy checking: Does a particular measurement add any new information, given the set of possible faults and what's been observed so far? (Is the student testing useful a hypothesis or is he just randomly making measurements?)

It is possible to apply these ideas to a tutor built upon NEOMYCIN. But imagine the difficulty of answering the question, "What if the patient didn't have a fever? What if the patient's fever were 110?" Imagine the complexity of the model of the body and the kind of computations that would be necessary to deal with any question of this form. The power of SOPHIE comes from the complete simulation model, which makes certain assumptions, of course, about how the circuit works and what the current environment is. In medicine we can't perform this hypothetical generation and question answering. We can't say everything that could be wrong. We have a model, but we don't have a basis for performing experiments for determining how the patient would behave under different environmental loads.

This is a subtle point. The problem arises not just because we have represented a small part of the space of possible diseases in NEOMYCIN. Diseases in medicine are different from faults in a circuit. Diseases in medicine are not faulty components, they are interactions that the "device" has with the world that cause the components to fail. You cannot possibly enumerate all of the world knowledge that could cause all the possible diseases that are or could be observed.

To give an example: Consider a disease like tennis elbow. Suppose someone plays tennis and gets a sore arm and he comes to you and says, "Something is wrong with my arm." You need to know to tell him to stop playing tennis. Just having a model of the body says nothing about tennis. You need to know all of the ways that people interact with the world. SOPHIE's form of electronic diagnosis has nothing to do with how the circuit interacts with the world. If the fault is caused by a room that is too warm or because somebody dropped the power regulator before they installed it, SOPHIE will fail. To restate, a disease in medicine is not a component fault, but what caused the fault in the world. [This might remind you of the difference between a procedural bug (DEBUGGY) and the problem-solving and learning process that generated this bug (Repair and Step Theory).]

There is a lot more to say about SOPHIE; read the paper in the *Intelligent Tutoring Systems*

book. It explains in detail the work of deKleer which came later, involving an expert system that could perform and describe the diagnostic process.

2.11. STEAMER

STEAMER teaches how to operate a steam plant in a ship (Hollan, et al., 1984). The use of graphics coupled to a mathematical simulation program and semantic network descriptions allows the program to provide a high-fidelity conceptual model of how the steam plant works. STEAMER is the work of Stevens, Forbus, and several people of the Naval Personnel and Research Development Center, including Jim Hollan and Mike Hutchins. The project was originated by a psychologist, Mike Williams, who had experience on a ship teaching and learning steam-propulsion plant operation.

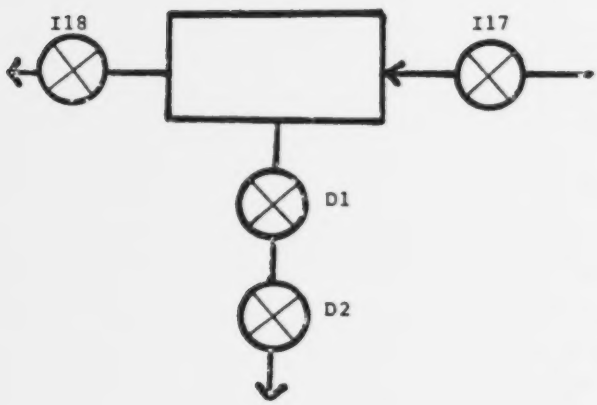
The project was designed to be put into the field as a prototype within a few years; today they are using STEAMER to train new recruits. In general, the military is excited about automating teaching because of their high turnover and the great deal of technical expertise to be learned in many different areas.

The major difficulty here is to understand the rationale for why the procedures for running the steam plant are correct, particularly to facilitate remembering and modifying them. Valves, pipes, and wires are arranged in a complicated jumble inside the ship. When a student is brought on board, it's very hard for him to understand how things are connected to one another. Graphics provide an easy way of retrieving the information. You can look at the lubricating system and you can look at the steam system. This graphics system is connected to the underlying simulation program, using the idea of active images or "active values" in the LOOPS programming system, developed at Xerox-Parc. The program sends a message to a particular concept or a unit, say one that corresponds to a valve. The valve instance is then responsible for setting itself on the screen to reflect its value.

Figure 2-14 shows part of the teaching problem in STEAMER. The procedure says, "Step 5, align the drain valves D1 and D2." Much later, it says, "Open valve I17." This is what you are supposed to do. The rationale for this procedure is not explicit—why should you do step 5 at all? Why should you do it before step 12? We want to teach the rationale as well as the procedure: Whenever steam is admitted to a chamber, you must align the drains first. If you don't, the water that is left in the chamber will mix with the steam, and high-energy water pellets will get thrown downstream.

STEAMER research included new work in explanation. Using the description of the current state of the plant, the program generates explanations of what is happening. For example, it can describe a feedback process. They've studied to some extent the problem of packaging the text and presenting overviews, as well as going through causal detail. There is some natural language work, but the research has emphasized causal modelling.

Operating Procedure = Order of Steps



5. Align Drain Valves D1 and D2.

12. Open Input Valve I17.

Why?

Figure 2-14: Schematic from STEAMER with steps in operational procedure

3. Conclusion

How well do these programs perform? Is anybody using them? Some of them have been tested, but there is no system that has in any significant way replaced the traditional methods of teaching. These are mainly pilot studies that demonstrate plausibility. STEAMER might be the closest to a system that has replaced some traditional teaching and maybe greatly augmented what can be taught. DEBUGGY has been used for thousands of students, but as data collection to verify the model, not interactively. MENO is being used at Yale, for teaching introductory programming. The WEST system was used as an exploratory system for a couple of months in an elementary school. SOPHIE was used experimentally for a few years over a computer network within the U.S. Recent programs by Anderson and Reiser are effectively teaching Lisp and Geometry (Anderson, et al., 1984b, Anderson, et al., 1985).

In short, education has not been turned upside down. The process of developing these ideas and getting the programs to be used is very slow. I believe that professional educators in the schools today, not the researchers who developed the programs, will be the people who eventually take these ideas and apply them.

So what is all this worth? How would we compare the state of CAI and ICAI today? There is a mixture of goals and accomplishments. I think if we did a very simple analysis of the best traditional programs today, we would say that they are much better than almost any of the ICAI tutoring systems. They are better when viewed in terms of being operating, portable programs that students can actually learn from. Many hours of work were put into designing CAI programs, with a lot of handcrafting of the design to make them useful. I know a physician at Stanford who has been cranking out CAI programs for probably 20 years. He felt embarrassed when he saw what we were doing. He felt that we were coming from another planet and believed that now all of his work was obsolete. I told him that was not true at all. He is teaching students today with his programs, and it might be ten years before we have anything to give him. That was in 1977. His programs still work, and they are teaching many students every year. ICAI is mainly a lot of promises still.

On the other hand, these promises are profound and changing not just how we think about teaching, but our understanding of what needs to be taught. We should remember that CAI systems don't have a systematic conceptual representation of what is being taught. The person who designs the traditional system is not provided with a language that allows him to write down what he knows in a structured, reusable way. Very little cumulative science or engineering is occurring. Each CAI author is building in his knowledge and not effectively sharing it with other teachers except to say, "Here is my program, you can learn from my example." The field will advance much more quickly if people could share their knowledge of learning and pedagogy, and develop formal theories. While WEST, WHY, GUIDON, and other ICAI programs take many years to develop, ICAI researchers can now say, "Here is my set of rules: Use them in your system." The idea of shared knowledge bases is something that excites me a great deal when I consider where we might be in ten or twenty years, after collecting and abstracting knowledge in tutoring and expert systems.

Acknowledgments

This paper is an edited version of a presentation given in Namur, Belgium in May, 1985, as part of the program for the International Professorship in Computer Science (Expert Systems), Universite de L'Etat, sponsored by the Belgian National Foundation for Scientific Research and IBM. I sincerely thank my friend Axel van Lamsweerde for his hospitality during my stay and for his arduous efforts to produce a transcript of this presentation. While the original transcript has been edited a great deal, this paper is intended to reflect the extemporaneous nature of an oral presentation, rather than a scholarly work. I have however added citations to bring the material up-to-date. As indicated in the text, much of the material was prepared by John Seely Brown and Richard R. Burton, or in collaboration with them. Much of what I know about Intelligent Tutoring Systems comes from working with John and Richard.

My research is sponsored in part by ONR (contract N00014-85-K-0305) and by a grant from the Josiah Macy, Jr. Foundation. Computational resources have been provided by the Sumex-Aim National Resource (NIH grant RR00785).

References

Anderson, J.R., Boyle, C.F., Farrell, R., and Reiser, B. *Cognitive principles in the design of computer tutors*, in *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, pages 2-10, Boulder, June, 1984.

Anderson, J.R., Farrell, R. and Sauers, R. Learning to program in LISP. *Cognitive Science*, April-June 1984, *8(2)*, 87-129.

Anderson, J.R., Boyle, C.F., Yost, G. *The geometry tutor*, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1-7, Los Angeles, August, 1985. Volume 1.

Brown, J. S. and Burton, R. B. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, April-June 1978, *2(2)*, 155-192.

Brown, J. S. *Process versus product--a perspective on tools for communal and informal electronic learning*, in *Education in the Electronic Age, proceedings of a conference sponsored by the Educational Broadcasting Corporation, WNET/Thirteen*, July, 1983.

Carbonell, J. R. *Mixed-initiative man-computer instructional dialogues*. Technical Report 1971, Bolt Beranek and Newman, 1970.

Carbonell, J. R., and Collins, A. M. *Natural semantics in artificial intelligence*, in *Proc. 3rd IJCAI, Stanford, CA*, pages 344-351, IJCAI, August, 1973.

Clancey, W. J. GUIDON. In Barr and Feigenbaum (editors), *The Handbook of Artificial Intelligence*, chapter Applications-oriented AI research: Educationpages 267-278. William Kaufmann, Inc., Los Altos, 1982.

Clancey, W.J. Qualitative student models. (To appear in the first *Annual Review of Computer Science*, Palo Alto: Annual Reviews, Inc., 1986.).

Clancey, W. J. and Letsinger, R. NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In Clancey, W. J. and Shortliffe, E. H. (editors), *Readings in Medical Artificial Intelligence: The First Decade*, pages 361-381. Addison-Wesley, Reading, 1984.

Collins, A. *Fragments of a theory of human plausible reasoning*, in *Proc. 2nd Conference on Theoretical Issues in Natural Language Processing*, pages 194-201, TINLAP, July, 1978.

Collins, A. and Brown, J.S. The computer as a tool for learning through reflection. In H. Mandl and A. Lesgold (editors), *Learning Issues for Intelligent Tutoring Systems*, . Springer, New York, In press.

Collins, A. and Stevens, A. L. *Goals and strategies of interactive teachers*. BBN Technical Report 4345, Bolt, Beranek, and Newman, 1980.

Genesereth, M. R. The role of plans in intelligent teaching systems. In D. Sleeman and J. S. Brown (editors), *Intelligent Tutoring Systems*, pages 137-155. Academic Press, New York, 1982.

Genesereth, M. R. *An overview of meta-level architecture*, in *Proceedings of The National Conference on Artificial Intelligence*, pages 119-124, August, 1983.

Goldstein, I.P. *The Computer as Coach: An Athletic Paradigm for Intellectual Education*. AI

Memo 389, AI Laboratory, Massachusetts Institute of Technology, 1977.

Goldstein, I. *Developing a computational representation for problem solving skills*, in *Proc. Carnegie-Mellon Conference on Problem Solving and Education: Issues in Teaching and Research*, October 9-10, 1978.

Goldstein, I.P. The genetic graph: a representation for the evolution of procedural knowledge. In D. Sleeman and J.S. Brown (editors), *Intelligent Tutoring Systems*, pages 51-77. Academic Press, London, 1982.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. STEAMER: An interactive inspectable simulation-based training system. *The AI Magazine*, 1984, *5(2)*, 15-27.

Johnson, W. Lewis, and Elliot Soloway. *Intention-Based Diagnosis of Programming Errors*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 162-168, Austin, TX, August, 1984.

Kolodner, J. L. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 1983, *7*, 243-280.

Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, Inc. 1980.

Schank, R. C. Failure-driven memory. *Cognition and Brain Theory*, 1981, *4(1)*, 41-60.

Sleeman, D. and Brown, J. S. (editors). *Intelligent Tutoring Systems*. New York: Academic Press 1982.

Soloway, E.M, Woolf, B., Rubin, E., Barth, P. *Meno-II: An intelligent tutoring system for novice programmers*, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 975-977, Vancouver, August, 1981.

Soloway, E. M. Rubin, E., Woolf, B., Bonar, J., and Johnson, W. L. *Meno-II: An AI-based programming tutor*. Technical Report CSD/RR No. 258, Yale University, December 1982.

VanLehn, K. *Human procedural skill acquisition: Theory, model, and psychological validation*, in *Proceedings of the National Conference on AI*, pages 420-423, Washington, D.C., August, 1983.

VanLehn, K., Brown, J. S., Greeno, J. Competitive argumentation in computational theories of cognition. In Kintsch, Miller, and Polson (editors), *Method and Tactics in Cognitive Science*, pages 235-262. Lawrence Erlbaum Associates, Hillsdale, NJ, 1984.

Wenger, E. AI and the communication of knowledge: an overview of intelligent teaching systems. To be published by Morgan-Kaufmann, Los Altos, CA.

Westcourt, K. T., Beard, M. and Gould, M. *Knowledge-based adaptive curriculum sequences for CAI: Application of a network representation.* Tech Report 288, Stanford University, Institute for Mathematical Studies in the Social Sciences, 1977.

Woolf, B. and McDonald, D.D. *Context-dependent transitions in tutoring discourse*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 355-361, Austin, August, 1984.

Personnel Analysis Division,
AF/MPXA
5C360, The Pentagon
Washington, DC 20330

Air Force Human Resources Lab
AFHRL/MPD
Brooks AFB, TX 78235

AFOSR,
Life Sciences Directorate
Bolling Air Force Base
Washington, DC 20332

Dr. Robert Ahlers
Code N711
Human Factors Laboratory
Naval Training Systems Center
Orlando, FL 32813

Dr. Ed Aiken
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. William E. Alley
AFHRL/MOT
Brooks AFB, TX 78235

Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Dr. Nancy S. Anderson
Department of Psychology
University of Maryland
College Park, MD 20742

Dr. Steve Andriole
George Mason University
School of Information
Technology & Engineering
4400 University Drive
Fairfax, VA 22030

Dr. John Annett
University of Warwick
Department of Psychology
Coventry CV4 7AJ
ENGLAND

Dr. Phipps Arabie
University of Illinois
Department of Psychology
603 E. Daniel St.
Champaign, IL 61820

Technical Director, ARI
5001 Eisenhower Avenue
Alexandria, VA 22333

Special Assistant for Projec
OASN(M&RA)
5D800, The Pentagon
Washington, DC 20350

Dr. Michael Atwood
ITT - Programming
1000 Oronoque Lane
Stratford, CT 06497

Dr. Patricia Baggett
University of Colorado
Department of Psychology
Box 345
Boulder, CO 80309

Dr. Eva L. Baker
UCLA Center for the Study
of Evaluation
145 Moore Hall
University of California
Los Angeles, CA 90024

Dr. James D. Baker
Director of Automation
Allen Corporation of America
401 Wythe Street
Alexandria, VA 22314

Dr. Meryl S. Baker
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Donald E. Bamber
Code 71
Navy Personnel R & D Center
San Diego, CA 92152-6800

Mr. J. Barber
HQS, Department of the Army
DAPE-ZBR
Washington, DC 20310

Capt. J. Jean Belanger
Training Development Division
Canadian Forces Training System
CFTSHQ, CFB Trenton
Astra, Ontario, K0K
CANADA

CDR Robert J. Biersner, USN
Naval Biodynamics Laboratory
P. O. Box 29407
New Orleans, LA 70189

Dr. Menucha Birenbaum
School of Education
Tel Aviv University
Tel Aviv, Ramat Aviv 69978
ISRAEL

Dr. Werner P. Birke
Personalstammamt der Bundeswehr
Kolner Strasse 262
D-5000 Koeln 90
FEDERAL REPUBLIC OF GERMANY

Dr. Gautam Biswas
Department of Computer Science
University of South Carolina
Columbia, SC 29208

Dr. John Black
College, Columbia Univ.
525 West 121st Street
New York, NY 10027

Dr. Arthur S. Blaiwes
Code N711
Naval Training Systems Center
Orlando, FL 32813

Dr. Robert Blanchard
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Jeff Bonar
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. J. C. Boudreaux
Center for Manufacturing
Engineering
National Bureau of Standards
Gaithersburg, MD 20899

Dr. Gordon H. Bower
Department of Psychology
Stanford University
Stanford, CA 94306

Dr. Richard Braby
NTSC Code 10
Orlando, FL 32751

Dr. Robert Breaux
Code N-095R
Naval Training Systems Cente
Orlando, FL 32813

Dr. John S. Brown
XEROX Palo Alto Research
Center
3333 Coyote Road
Palo Alto, CA 94304

Dr. Bruce Buchanan
Computer Science Department
Stanford University
Stanford, CA 94305

Dr. Howard Burrows
National Institute of Health
Bldg. 10, Room 5C-106
Bethesda, MD 20892

Dr. Patricia A. Butler
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. Tom Cafferty
Dept. of Psychology
University of South Carolina
Columbia, SC 29208

Dr. Robert Calfee
School of Education
Stanford University
Stanford, CA 94305

Joanne Capper
Center for Research into Practice
1718 Connecticut Ave., N.W.
Washington, DC 20009

Dr. Jaime Carbonell
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Mr. James W. Carey
Commandant (G-PTE)
U.S. Coast Guard
2100 Second Street, S.W.
Washington, DC 20593

Dr. Susan Carey
Harvard Graduate School of
Education
337 Gutman Library
Appian Way
Cambridge, MA 02138

Dr. Pat Carpenter
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Prof. John W. Carr III
Department of Computer Science
Moore School of
Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104

Dr. Robert Carroll
OP 01B7
Washington, DC 20370

LCDR Robert Carter
Office of the Chief
of Naval Operations
OP-01B
Pentagon
Washington, DC 20350-2000

Dr. Fred Chang
Navy Personnel R&D Center
Code 51
San Diego, CA 92152-6800

Dr. Alphonse Chapanis
8415 Bellona Lane
Suite 210
Buxton Towers
Baltimore, MD 21204

Dr. Davida Charney
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Eugene Charniak
Brown University
Computer Science Department
Providence, RI 02912

Dr. Paul R. Chatelier
OUSDRE
Pentagon
Washington, DC 20350-2000

Dr. Michelene Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Mr. Raymond E. Christal
AFHRL/MOE
Brooks AFB, TX 78235

Dr. Yee-Yeen Chu
Perceptronics, Inc.
21111 Erwin Street
Woodland Hills, CA 91367-371

Dr. William Clancey
Stanford University
Knowledge Systems Laboratory
701 Welch Road, Bldg. C
Palo Alto, CA 94304

Chief of Naval Education
and Training
Liaison Office
Air Force Human Resource Lab
Operations Training Division
Williams AFB, AZ 85224

Assistant Chief of Staff
for Research, Development,
Test, and Evaluation
Naval Education and
Training Command (N-5)
NAS Pensacola, FL 32508

Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. John J. Collins
Director, Field Research
Office, Orlando
NPRDC Liaison Officer
NTSC Orlando, FL 32813

Dr. Stanley Collyer
Office of Naval Technology
Code 222
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Lynn A. Cooper
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Dr. Meredith P. Crawford
American Psychological Association
Office of Educational Affairs
1200 17th Street, N.W.
Washington, DC 20036

Dr. Hans Crombag
University of Leyden
Education Research Center
Boerhaavelaan 2
2334 EN Leyden
The NETHERLANDS

LT Judy Crookshanks
Chief of Naval Operations
OP-112G5
Washington, DC 20370-2000

Dr. Kenneth B. Cross
Anacapa Sciences, Inc.
P.O. Drawer Q
Santa Barbara, CA 93102

Dr. Mary Cross
Department of Education
Adult Literacy Initiative
Room 4145
400 Maryland Avenue, SW
Washington, DC 20202

CTB/McGraw-Hill Library
2500 Garden Road
Monterey, CA 93940

CAPT P. Michael Curran
Office of Naval Research
800 N. Quincy St.
Code 125
Arlington, VA 22217-5000

Dr. Cary Czichon
Intelligent Instructional Sy
Texas Instruments AI Lab
P.O. Box 660245
Dallas, TX 75266

Bryan Dallman
AFHRL/LRT
Lowry AFB, CO 80230

LT John Deaton
ONR Code 125
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Natalie Dehn
Department of Computer and
Information Science
University of Oregon
Eugene, OR 97403

Dr. Gerald F. DeJong
Artificial Intelligence Grou
Coordinated Science Laborato
University of Illinois
Urbana, IL 61801

Goery Delacote
Directeur de L'informatique
Scientifique et Technique
CNRS
15, Quai Anatole France
75700 Paris FRANCE

Mr. Robert Denton
AFMPC/MPCYPR
Randolph AFB, TX 78150

Mr. Paul DiRenzo
Commandant of the Marine Corps
Code LBC-4
Washington, DC 20380

Dr. R. K. Dismukes
Associate Director for Life Sciences
AFOSR
Bolling AFB
Washington, DC 20332

Defense Technical
Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC
(12 Copies)

Dr. Thomas M. Duffy
Communications Design Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Barbara Eason
Military Educator's
Resource Network
InterAmerica Research Associates
1555 Wilson Blvd
Arlington, VA 22209

Edward E. Eddowes
CNATRA N301
Naval Air Station
Corpus Christi, TX 78419

Dr. John Ellis
Navy Personnel R&D Center
San Diego, CA 92252

Dr. Richard Elster
Deputy Assistant Secretary
of the Navy (Manpower)
OASN (M&RA)
Department of the Navy
Washington, DC 20350-1000

Dr. Susan Embretson
University of Kansas
Psychology Department
426 Fraser
Lawrence, KS 66045

Dr. Randy Engle
Department of Psychology
University of South Carolina
Columbia, SC 29208

Lt. Col Rich Entlich
HQ, Department of the Army
OCSA(DACS-DPM)
Washington, DC 20310

Dr. Susan Epstein
Hunter College
144 S. Mountain Avenue
Montclair, NJ 07042

Dr. William Epstein
University of Wisconsin
W. J. Brogden Psychology Bld
1202 W. Johnson Street
Madison, WI 53706

ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

Dr. K. Anders Ericsson
University of Colorado
Department of Psychology
Boulder, CO 80309

Dr. Edward Esty
Department of Education, OER
Room 717D
1200 19th St., NW
Washington, DC 20208

Dr. Beatrice J. Farr
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Marshall J. Farr
2520 North Vernon Street
Arlington, VA 22207

Dr. Pat Federico
Code 511
NPRDC
San Diego, CA 92152-6800

Dr. Jerome A. Feldman
University of Rochester
Computer Science Department
Rochester, NY 14627

Dr. Paul Feltovich
Southern Illinois University
School of Medicine
Medical Education Department
P.O. Box 3926
Springfield, IL 62708

Mr. Wallace Feurzeig
Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

Dr. Craig I. Fields
ARPA
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Gerhard Fischer
University of Colorado
Department of Computer Science
Boulder, CO 80309

J. D. Fletcher
9931 Corsica Street
Vienna VA 22180

Dr. Linda Flower
Carnegie-Mellon University
Department of English
Pittsburgh, PA 15213

Dr. Kenneth D. Forbus
University of Illinois
Department of Computer Science
1304 West Springfield Avenue
Urbana, IL 61801

Dr. Barbara A. Fox
University of Colorado
Department of Linguistics
Boulder, CO 80309

Dr. Carl H. Frederiksen
McGill University
3700 McTavish Street
Montreal, Quebec H3A 1Y2
CANADA

Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

Dr. Norman Frederiksen
Educational Testing Service
Princeton, NJ 08541

Dr. Alfred R. Fregly
AFOSR/NL
Bolling AFB, DC 20332

Dr. Bob Frey
Commandant (G-P-1/2)
USCG HQ
Washington, DC 20593

Dr. Alinda Friedman
Department of Psychology
University of Alberta
Edmonton, Alberta
CANADA T6G 2E9

Julie A. Gadsden
Information Technology
Applications Division
Admiralty Research Establish
Portsdown, Portsmouth PO6 4A
UNITED KINGDOM

Dr. R. Edward Geiselman
Department of Psychology
University of California
Los Angeles, CA 90024

Dr. Michael Genesereth
Stanford University
Computer Science Department
Stanford, CA 94305

Dr. Dedre Gentner
University of Illinois
Department of Psychology
603 E. Daniel St.
Champaign, IL 61820

Dr. Robert Glaser
Learning Research
& Development Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

Dr. Arthur M. Glenberg
University of Wisconsin
W. J. Brogden Psychology Bldg.
1202 W. Johnson Street
Madison, WI 53706

Dr. Marvin D. Glock
13 Stone Hall
Cornell University
Ithaca, NY 14853

Dr. Gene L. Gloye
Office of Naval Research
Detachment
1030 E. Green Street
Pasadena, CA 91106-2485

Dr. Sam Glucksberg
Department of Psychology
Princeton University
Princeton, NJ 08540

Dr. Joseph Goguen
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dr. Sherrie Gott
AFHRL/MODJ
Brooks AFB, TX 78235

Jordan Grafman, Ph.D.
2021 Lyttonsville Road
Silver Spring, MD 20910

Dr. Richard H. Granger
Department of Computer Science
University of California, Irvine
Irvine, CA 92717

Dr. Wayne Gray
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. James G. Greeno
University of California
Berkeley, CA 94720

H. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

Dipl. Pad. Michael W. Habon
Universitat Dusseldorf
Erziehungswissenschaftliches
Universitatsstr. 1
D-4000 Dusseldorf 1
WEST GERMANY

Prof. Edward Haertel
School of Education
Stanford University
Stanford, CA 94305

Dr. Henry M. Halff
Halff Resources, Inc.
4918 33rd Road, North
Arlington, VA 22207

Dr. Ronald K. Hambleton
Prof. of Education & Psychol
University of Massachusetts
at Amherst
Hills House
Amherst, MA 01003

Dr. Cheryl Hamel
NTSC
Orlando, FL 32813

Dr. Bruce W. Hamill
Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20707

Dr. Ray Hannapel
Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

Stevan Harnad
Editor, The Behavioral and
Brain Sciences
20 Nassau Street, Suite 240
Princeton, NJ 08540

Mr. William Hartung
PEAM Product Manager
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Wayne Harvey
SRI International
333 Ravenswood Ave.
Room B-S324
Menlo Park, CA 94025

Dr. Reid Hastie
Northwestern University
Department of Psychology
Evanston, IL 60201

Prof. John R. Hayes
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Barbara Hayes-Roth
Department of Computer Science
Stanford University
Stanford, CA 95305

Dr. Frederick Hayes-Roth
Teknowledge
525 University Ave.
Palo Alto, CA 94301

Dr. Joan I. Heller
505 Haddon Road
Oakland, CA 94606

Dr. James Hendler
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Dr. Jim Hollan
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92093

Dr. Melissa Holland
Army Research Institute for
Behavioral and Social Sci
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Robert W. Holt
Department of Psychology
George Mason University
4400 University Drive
Fairfax, VA 22030

Dr. Keith Holyoak
University of Michigan
Human Performance Center
330 Packard Road
Ann Arbor, MI 48109

Prof. Lutz F. Hornke
Institut fur Psychologie
RWTH Aachen
Jaegerstrasse 17/19
D-5100 Aachen
WEST GERMANY

Mr. Dick Hoshaw
OP-135
Arlington Annex
Room 2834
Washington, DC 20350

Dr. James Howard
Dept. of Psychology
Human Performance Laboratory
Catholic University of
America
Washington, DC 20064

Dr. Steven Hunka
Department of Education
University of Alberta
Edmonton, Alberta
CANADA

Dr. Earl Hunt
Department of Psychology
University of Washington
Seattle, WA 98105

Dr. Ed Hutchins
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92093

Dr. Dillon Inouye
WICAT Education Institute
Provo, UT 84057

Dr. Alice Isen
Department of Psychology
University of Maryland
Catonsville, MD 21228

Dr. Zachary Jacobson
Bureau of Management Consulting
365 Laurier Avenue West
Ottawa, Ontario K1A 0S5
CANADA

Dr. Robert Jannarone
Department of Psychology
University of South Carolina
Columbia, SC 29208

Dr. Claude Janvier
Directeur, CIRADE
Universite' du Quebec a Montreal
Montreal, Quebec H3C 3P8
CANADA

COL Dennis W. Jarvi
Commander
AFHRL
Brooks AFB, TX 78235-5601

Dr. Robin Jeffries
Hewlett-Packard Laboratories
P.O. Box 10490
Palo Alto, CA 94303-0971

Dr. Robert Jernigan
Decision Resource Systems
5595 Vantage Point Road
Columbia, MD 21044

Margaret Jerome
c/o Dr. Peter Chandler
83, The Drive
Hove
Sussex
UNITED KINGDOM

Dr. Joseph E. Johnson
Assistant Dean for
Graduate Studies
College of Science and Mathe
University of South Carolina
Columbia, SC 29208

Dr. Richard Johnson
Boise State University
Simplot/Micron
Technology Center
Boise, ID 83725

CDR Tom Jones
ONR Code 125
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Douglas A. Jones
P.O. Box 6640
Lawrenceville
NJ 08648

Col. Dominique Jouslin de No
Etat-Major de l'Armee de Ter
Centre de Relations Humaines
3 Avenue Octave Greard
75007 Paris
FRANCE

Dr. Marcel Just
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Milton S. Katz
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Scott Kelso
Haskins Laboratories,
270 Crown Street
New Haven, CT 06510

Dr. Dennis Kibler
University of California
Department of Information
and Computer Science
Irvine, CA 92717

Dr. David Kieras
University of Michigan
Technical Communication
College of Engineering
1223 E. Engineering Building
Ann Arbor, MI 48109

Dr. Peter Kincaid
Training Analysis
& Evaluation Group
Department of the Navy
Orlando, FL 32813

Dr. Walter Kintsch
Department of Psychology
University of Colorado
Campus Box 345
Boulder, CO 80302

Dr. Paula Kirk
Oakridge Associated Universities
University Programs Division
P.O. Box 117
Oakridge, TN 37831-0117

Dr. David Klahr
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Mazie Knerr
Program Manager
Training Research Division
HumRRO
1100 S. Washington
Alexandria, VA 22314

Dr. Janet L. Kolodner
Georgia Institute of Technology
School of Information
& Computer Science
Atlanta, GA 30332

Dr. Stephen Kosslyn
Harvard University
1236 William James Hall
33 Kirkland St.
Cambridge, MA 02138

Dr. Kenneth Kotovsky
Department of Psychology
Community College of
Allegheny County
800 Allegheny Avenue
Pittsburgh, PA 15233

Dr. David H. Krantz
2 Washington Square Village
Apt. # 15J
New York, NY 10012

Dr. Benjamin Kuipers
University of Texas at Austi
Department of Computer Scien
T.S. Painter Hall 3.28
Austin, Texas 78712

Dr. David R. Lambert
Naval Ocean Systems Center
Code 441T
271 Catalina Boulevard
San Diego, CA 92152-6800

Dr. Pat Langley
University of California
Department of Information
and Computer Science
Irvine, CA 92717

M. Diane Langston
Communications Design Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Jill Larkin
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Dr. Robert Lawler
Information Sciences, FRL
GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254

Dr. Paul E. Lehner
PAR Technology Corp.
7926 Jones Branch Drive
Suite 170
McLean, VA 22102

Dr. Alan M. Lesgold
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Alan Leshner
Deputy Division Director
Behavioral and Neural Sciences
National Science Foundation
1800 G Street
Washington, DC 20550

Dr. Jim Levin
University of California
Laboratory for Comparative
Human Cognition
D003A
La Jolla, CA 92093

Dr. John Levine
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Michael Levine
Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

Dr. Charles Lewis
Faculteit Sociale Wetenschappen
Rijksuniversiteit Groningen
Oude Boteringestraat 23
9712GC Groningen
The NETHERLANDS

Dr. Clayton Lewis
University of Colorado
Department of Computer Science
Campus Box 430
Boulder, CO 80309

Library
Naval War College
Newport, RI 02940

Library
Naval Training Systems Cente
Orlando, FL 32813

Dr. Charlotte Linde
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dr. Marcia C. Linn
Lawrence Hall of Science
University of California
Berkeley, CA 94720

Dr. Robert Linn
College of Education
University of Illinois
Urbana, IL 61801

Dr. Don Lyon
P. O. Box 44
Higley, AZ 85236

Dr. Jane Malin
Mail Code SR 111
NASA Johnson Space Center
Houston, TX 77058

Dr. William L. Maloy
Chief of Naval Education
and Training
Naval Air Station
Pensacola, FL 32508

Dr. Sandra P. Marshall
Dept. of Psychology
San Diego State University
San Diego, CA 92182

Dr. Manton M. Matthews
Department of Computer Scien
University of South Carolina
Columbia, SC 29208

Dr. Richard E. Mayer
Department of Psychology
University of California
Santa Barbara, CA 93106

Dr. James McBride
Psychological Corporation
c/o Harcourt, Brace,
Javanovich Inc.
1250 West 6th Street
San Diego, CA 92101

Dr. David J. McGuinness
Gallaudet College
800 Florida Avenue, N.E.
Washington, DC 20002

Dr. Kathleen McKeown
Columbia University
Department of Computer Science
New York, NY 10027

Dr. Joe McLachlan
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. James McMichael
Assistant for MPT Research,
Development, and Studies
OP 01B7
Washington, DC 20370

Dr. Barbara Means
Human Resources
Research Organization
1100 South Washington
Alexandria, VA 22314

Dr. Douglas L. Medin
Department of Psychology
University of Illinois
603 E. Daniel Street
Champaign, IL 61820

Dr. Arthur Melmed
U. S. Department of Education
724 Brown
Washington, DC 20208

Dr. Jose Mestre
Department of Physics
Hasbrouck Laboratory
University of Massachusetts
Amherst, MA 01003

Dr. Al Meyrowitz
Office of Naval Research
Code 1133
800 N. Quincy
Arlington, VA 22217-5000

Dr. Ryszard S. Michalski
University of Illinois
Department of Computer Scien
1304 West Springfield Avenue
Urbana, IL 61801

Prof. D. Michie
The Turing Institute
36 North Hanover Street
Glasgow G1 2AD, Scotland
UNITED KINGDOM

Dr. George A. Miller
Department of Psychology
Green Hall
Princeton University
Princeton, NJ 08540

Dr. James R. Miller
MCC
9430 Research Blvd.
Echelon Building #1, Suite 2
Austin, TX 78759

Dr. Mark Miller
Computer*Thought Corporation
1721 West Plano Parkway
Plano, TX 75075

Dr. Andrew R. Molnar
Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

Dr. William Montague
NPRDC Code 13
San Diego, CA 92152-6800

Dr. Tom Moran
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Dr. Melvyn Moy
Navy Personnel R & D Center
San Diego, CA 92152-6800

Dr. Allen Munro
Behavioral Technology
Laboratories - USC
1845 S. Elena Ave., 4th Floor
Redondo Beach, CA 90277

Director,
Decision Support
Systems Division, NMPC
Naval Military Personnel Command
N-164
Washington, DC 20370

Director,
Distribution Department, NMPC
N-4
Washington, DC 20370

Director,
Overseas Duty Support
Program, NMPC
N-62
Washington, DC 20370

Head, HRM Operations Branch,
NMPC
N-62F
Washington, DC 20370

Assistant for Evaluation,
Analysis, and MIS, NMPC
N-6C
Washington, DC 20370

Spec. Asst. for Research, Experi-
mental & Academic Programs,
NTTC (Code 016)
NAS Memphis (75)
Millington, TN 38054

Director,
Research & Analysis Div.,
NAVCRUITCOM Code 22
4015 Wilson Blvd.
Arlington, VA 22203

Technical Director
Navy Health Research Center
P.O. Box 85122
San Diego, CA 92138

Leadership Management Educat
and Training Project Offi
Naval Medical Command
Code 05C
Washington, DC 20372

Technical Director,
Navy Health Research Ctr.
P.O. Box 85122
San Diego, CA 92138

Dr. T. Niblett
The Turing Institute
36 North Hanover Street
Glasgow G1 2AD, Scotland
UNITED KINGDOM

Dr. Richard E. Nisbett
University of Michigan
Institute for Social Researc
Room 5261
Ann Arbor, MI 48109

Dr. Mary Jo Nissen
University of Minnesota
N218 Elliott Hall
Minneapolis, MN 55455

Dr. Donald A. Norman
Institute for Cognitive Scie
University of California
La Jolla, CA 92093

Director, Training Laborator
NPRDC (Code 05)
San Diego, CA 92152-6800

Director, Manpower and Perso
Laboratory,
NPRDC (Code 06)
San Diego, CA 92152-6800

Director, Human Factors
& Organizational Systems
NPRDC (Code 07)
San Diego, CA 92152-6800

Fleet Support Office,
NPRDC (Code 301)
San Diego, CA 92152-6800

Library, NPRDC
Code P201L
San Diego, CA 92152-6800

Commanding Officer,
Naval Research Laboratory
Code 2627
Washington, DC 20390

Dr. Harold F. O'Neil, Jr.
School of Education - WPH 801
Department of Educational
Psychology & Technology
University of Southern California
Los Angeles, CA 90089-0031

Dr. Stellan Ohlsson
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Director, Research Programs,
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1133
800 N. Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142
800 N. Quincy St.
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142EP
800 N. Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142PT
800 N. Quincy Street
Arlington, VA 22217-5000
(6 Copies)

Director, Technology Programs,
Office of Naval Research
Code 12
800 North Quincy Street
Arlington, VA 22217-5000

Special Assistant for Marine
Corps Matters,
ONR Code 00MC
800 N. Quincy St.
Arlington, VA 22217-5000

Assistant for Long Range
Requirements,
CNO Executive Panel
OP 00K
2000 North Beauregard Street
Alexandria, VA 22311

Assistant for MPT Research,
Development and Studies
OP 01B7
Washington, DC 20370

Dr. Judith Orasanu
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Jesse Orlansky
Institute for Defense Analys
1801 N. Beauregard St.
Alexandria, VA 22311

Prof. Seymour Papert
20C-109
Massachusetts Institute
of Technology
Cambridge, MA 02139

CDR R. T. Parlette
Chief of Naval Operations
OP-112G
Washington, DC 20370-2000

Dr. James Paulson
Department of Psychology
Portland State University
P.O. Box 751
Portland, OR 97207

Dr. Douglas Pearse
DCIEM
Box 2000
Downsview, Ontario
CANADA

Dr. Virginia E. Pendergrass
Code 711
Naval Training Systems Center
Orlando, FL 32813-7100

Dr. Robert Penn
NPRDC
San Diego, CA 92152-6800

Dr. Nancy Pennington
University of Chicago
Graduate School of Business
1101 E. 58th St.
Chicago, IL 60637

Military Assistant for Training and
Personnel Technology,
OUSD (R & E)
Room 3D129, The Pentagon
Washington, DC 20301-3080

Dr. Ray Perez
ARI (PERI-II)
5001 Eisenhower Avenue
Alexandria, VA 2233

Dr. David N. Perkins
Educational Technology Center
337 Gutman Library
Appian Way
Cambridge, MA 02138

LCDR Frank C. Petho, MSC, USN
CNATRA Code N36, Bldg. 1
NAS
Corpus Christi, TX 78419

Administrative Sciences Department,
Naval Postgraduate School
Monterey, CA 93940

Department of Computer Science,
Naval Postgraduate School
Monterey, CA 93940

Department of Operations Research,
Naval Postgraduate School
Monterey, CA 93940

Dr. Tjeerd Plomp
Twente University of Technol
Department of Education
P.O. Box 217
7500 AE ENSCHEDE

THE NETHERLANDS

Dr. Martha Polson
Department of Psychology
Campus Box 346
University of Colorado
Boulder, CO 80309

Dr. Peter Polson
University of Colorado
Department of Psychology
Boulder, CO 80309

Dr. Steven E. Poltrock
MCC
9430 Research Blvd.
Echelon Bldg #1
Austin, TX 78759-6509

Dr. Harry E. Pople
University of Pittsburgh
Decision Systems Laboratory
1360 Scaife Hall
Pittsburgh, PA 15261

Dr. Mary C. Potter
Department of Psychology
MIT (E-10-032)
Cambridge, MA 02139

Dr. Joseph Psotka
ATTN: PERI-1C
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

Dr. Roy Rada
National Library of Medicine
Bethesda, MD 20894

Dr. Lynne Reder
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Brian Reiser
Department of Psychology
Princeton University
Princeton, NJ 08544

Dr. James A. Reggia
University of Maryland
School of Medicine
Department of Neurology
22 South Greene Street
Baltimore, MD 21201

CDR Karen Reider
Naval School of Health Sciences
National Naval Medical Center
Bldg. 141
Washington, DC 20814

Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

Dr. Lauren Resnick
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Dr. Mary S. Riley
Program in Cognitive Science
Center for Human Information
Processing
University of California
La Jolla, CA 92093

William Rizzo
Code 712
Naval Training Systems Center
Orlando, FL 32813

Dr. Linda G. Roberts
Science, Education, and
Transportation Program
Office of Technology Assessment
Congress of the United States
Washington, DC 20510

Dr. Andrew M. Rose
American Institutes
for Research
1055 Thomas Jefferson St., NW
Washington, DC 20007

Dr. William B. Rouse
Search Technology, Inc.
25-b Technology Park/Atlanta
Norcross, GA 30092

Dr. Donald Rubin
Statistics Department
Science Center, Room 608
1 Oxford Street
Harvard University
Cambridge, MA 02138

Dr. David Rumelhart
Center for Human
Information Processing
Univ. of California
La Jolla, CA 92093

Ms. Riitta Ruotsalainen
General Headquarters
Training Section
Military Psychology Office
PL 919
SF-00101 Helsinki 10, FINLAN

Dr. Michael J. Samet
Perceptronics, Inc
6271 Variel Avenue
Woodland Hills, CA 91364

Dr. Roger Schank
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

Dr. W. L. Scherlis
Dept. of Computer Science
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Mrs. Birgitte Schneidelbach
Forsvarets Center for Leders
Christianshavns Voldgade 8
1424 Kobenhavn K
DENMARK

Dr. Walter Schneider
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

Dr. Alan H. Schoenfeld
University of California
Department of Education
Berkeley, CA 94720

Dr. Janet Schofield
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Marc Sebrechts
Department of Psychology
Wesleyan University
Middletown, CT 06475

Dr. Judith Segal
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. Robert J. Seidel
US Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

Dr. Ramsay W. Selden
Assessment Center
CCSSO
Suite 379
400 N. Capitol, NW
Washington, DC 20001

Dr. W. Steve Sellman
OASD(MRA&L)
2B269 The Pentagon
Washington, DC 20301

Dr. Sylvia A. S. Shafto
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. T. B. Sheridan
Dept. of Mechanical Engineering
MIT
Cambridge, MA 02139

Dr. Ben Shneiderman
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Dr. Ted Shortliffe
Computer Science Department
Stanford University
Stanford, CA 94305

Dr. Lee Shulman
Stanford University
1040 Cathcart Way
Stanford, CA 94305

Dr. Randall Shumaker
Naval Research Laboratory
Code 7510
4555 Overlook Avenue, S.W.
Washington, DC 20375-5000

Dr. Robert S. Siegler
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Herbert A. Simon
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

LTCOL Robert Simpson
Defense Advanced Research
Projects Administration
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Zita M Simutis
Instructional Technology
Systems Area
ARI
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. H. Wallace Sinaiko
Manpower Research
and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

Dr. Derek Sleeman
Stanford University
School of Education
Stanford, CA 94305

Dr. Charles F. Smith
North Carolina State Univers
Department of Statistics
Raleigh, NC 27695-8703

Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. Linda B. Smith
Department of Psychology
Indiana University
Bloomington, IN 47405

Dr. Alfred F. Smode
Senior Scientist
Code 07A
Naval Training Systems Center
Orlando, FL 32813

Dr. Richard E. Snow
Department of Psychology
Stanford University
Stanford, CA 94306

Dr. Elliot Soloway
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Kathryn T. Spoehr
Brown University
Department of Psychology
Providence, RI 02912

James J. Staszewski
Research Associate
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Marian Stearns
SRI International
333 Ravenswood Ave.
Room B-S324
Menlo Park, CA 94025

Dr. Robert Sternberg
Department of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

Dr. Paul J. Sticha
Senior Staff Scientist
Training Research Division
HumRRO
1100 S. Washington
Alexandria, VA 22314

Dr. Thomas Sticht
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. David Stone
KAJ Software, Inc.
3420 East Shea Blvd.
Suite 161
Phoenix, AZ 85028

Cdr Michael Suman, PD 303
Naval Training Systems Cente
Code N51, Comptroller
Orlando, FL 32813

Dr. Hariharan Swaminathan
Laboratory of Psychometric a
Evaluation Research
School of Education
University of Massachusetts
Amherst, MA 01003

Mr. Brad Sympson
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. John Tangney
AFOSR/NL
Bolling AFB, DC 20332

Dr. Kikumi Tatsuoka
CERL
252 Engineering Research
Laboratory
Urbana, IL 61801

Dr. Martin M. Taylor
DCIEM
Box 2000
Downsview, Ontario
CANADA

Dr. Perry W. Thorndyke
FMC Corporation
Central Engineering Labs
1185 Coleman Avenue, Box 580
Santa Clara, CA 95052

Major Jack Thorpe
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Douglas Towne
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

Dr. Amos Tversky
Stanford University
Dept. of Psychology
Stanford, CA 94305

Dr. James Tweeddale
Technical Director
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Paul Twohig
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Headquarters, U. S. Marine Corps
Code MPI-20
Washington, DC 20380

Dr. Kurt Van Lehn
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Beth Warren
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. David J. Weiss
N660 Elliott Hall
University of Minnesota
75 E. River Road
Minneapolis, MN 55455

Roger Weissinger-Baylon
Department of Administrative
Sciences
Naval Postgraduate School
Monterey, CA 93940

Dr. Donald Weitzman
MITRE
1820 Dolley Madison Blvd.
MacLean, VA 22102

Dr. Keith T. Wescourt
FMC Corporation
Central Engineering Labs
1185 Coleman Ave., Box 580
Santa Clara, CA 95052

Dr. Douglas Wetzel
Code 12
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Barbara White
Bolt Beranek & Newman, Inc.
10 Moulton Street
Cambridge, MA 02238

LCDR Cory deGroot Whitehead
Chief of Naval Operations
OP-112G1
Washington, DC 20370-2000

Dr. Michael Williams
IntelliCorp
1975 El Camino Real West
Mountain View, CA 94040-2216

Dr. Hilda Wing
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

A. E. Winterbauer
Research Associate
Electronics Division
Denver Research Institute
University Park
Denver, CO 80208-0454

Dr. Robert A. Wisher
U.S. Army Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Martin F. Wiskoff
Navy Personnel R & D Center
San Diego, CA 92152-6800

Dr. Merlin C. Wittrock
Graduate School of Education
UCLA
Los Angeles, CA 90024

Mr. John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Wallace Wulfeck, III
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Joe Yasatuke
AFHRL/LRT
Lowry AFB, CO 80230

Dr. Masoud Yazdani
Dept. of Computer Science
University of Exeter
Exeter EX4 4QL
Devon, ENGLAND

Major Frank Yohannan, USMC
Headquarters, Marine Corps
(Code MPI-20)
Washington, DC 20380

Mr. Carl York
System Development Foundation
181 Lytton Avenue
Suite 210
Palo Alto, CA 94301

Dr. Joseph L. Young
Memory & Cognitive
Processes
National Science Foundation
Washington, DC 20550

Dr. Steven Zornetzer
Office of Naval Research
Code 1140
800 N. Quincy St.
Arlington, VA 22217-5000

Dr. Michael J. Zyda
Naval Postgraduate School
Code 52CK
Monterey, CA 93943-5100